X68030 Inside/Out

桑野雅彦 *Masahiko Kuwano*●著

SOFT BANK

Cover Photo=Fumio SAITOU
Cover Design=Masaki KATSUMATA

X68030
Inside/Out

桒野雅彦 *Masahiko Kuwano*●著

まえがき

初代機以来，完全互換を維持してきたX68000シリーズも，X68000XVI，Compact XVI，X68030となるなど，X68000シリーズも高速化，高機能化が図られるようになりました。特にX68030は本格的な32ビットCPUを採用したことから，将来が楽しみな機種といえるでしょう。

これらの機種の追加にともない，内部のレジスタやコネクタなどにも若干の追加や変更が行われていますが，初代機をベースとして執筆されている『Inside X68000』（ソフトバンク刊）では，これらの情報については当然のことながら触れていません。

また，『Inside X68000』では，基本的にシャープからの公開情報に基づいて個々の機能について説明していましたが，アプリケーションソフトによっては非公開となっているビットの意味を独自に解析して使用している例も見られます。

本書では，X68030に対応したシャープからの追加情報に加え，X68000シリーズにおいて非公開情報となっている部分や，利用価値が高いにもかかわらず，具体的なタイミングがわからなかったような部分について，できるだけ実際の動作波形を示しながら説明することにしました。

また，X68030ではCPUにキャッシュメモリが搭載されたほか，MC68030と差し替えることで，UNIXなど仮想記憶をサポートしたOSを動かすことができます。今後、MC68040などを使ったX68000シリーズが登場する可能性も考えられるため，MC68020からMC68040までのM68000系CPUのキャッシュメモリとMMU機能についての説明も追加しました。

ともすれば，CPUの名称やクロック周波数ばかりが先行し，使い心地やエンドユーザによるプログラミングを考慮しない，閉鎖的なものが幅をきかせるなか，豊富な機能のすべてがユーザに公開された，希有なコンピュータであるX68000/X68030シリーズをさらに使い込むために本書を役立てていただけることを願ってやみません。

1994年1月　　桒野雅彦

CONTENTS

X68000と X68030の違い

X68030はX68000シリーズとの互換性を維持しつつ処理速度を向上したモデルですが，細かい部分まで見ていくと，完全に互換のとれていない部分もあります。ここでは，X68000とX68030の違いについてまとめてみました。

1 X68000XVIからX68030への主な変更点

X68030は，基本的にはX68000をベースにCPUをMC68HC000からMC68EC030に交換して高速化を図ったものであり，大きな機能追加などは行われていません。しかし，細かな点まで見ていくといくつかの違いが見つかります。違いの多くは周辺機器やソフトウェアにとっては影響のないものですが，コネクタの変更や数値演算プロセッサの扱いの変更など，注意を要するものもあります。ここでは，X68030で変更された点をまとめてみました。なお，比較の対象は特に断らないかぎり，X68000XVIとします。

X68030の主な変更点は，次のようになります。

①CPU

②ASIC（専用LSI）

③内部ブロック

④数値演算プロセッサ

⑤ROM

⑥メモリ増設方法
⑦SCSI IDの設定方法
⑧コネクタの廃止，形状／信号
⑨拡張スロットの信号や動作タイミング

これらのうち，②から④までは，①のCPUの変更によって連鎖的に変更されることになったものです。

①CPUの変更

X68000のCPUは，モトローラのM68000ファミリーのCPUとしては最も古いMC68000の低消費電力版であるMC68HC000（相当）でしたが，X68030ではM68000ファミリーの4代目にあたるMC68030からMMUを省略したMC68EC030を使用しています。また，クロック周波数も初代以来SUPERまでが10MHz，XVIとCompact XVIが16MHzでしたが，X68030では25MHzまで引き上げられています。

②ASICの変更

MC68030は，MC68000のアーキテクチャを踏襲しつつ飛躍的に性能を向上させたCPUです。MC68000の外部のデータバス幅が16ビットであるのに対し，MC68030では32ビットと拡張されているほか，高速化のため，基本的なバスアクセスのタイミングにも大幅な変更が行われています。このため，CPU動作と密接に関係しているシステムコントローラとメモリコントローラが作り直されました。

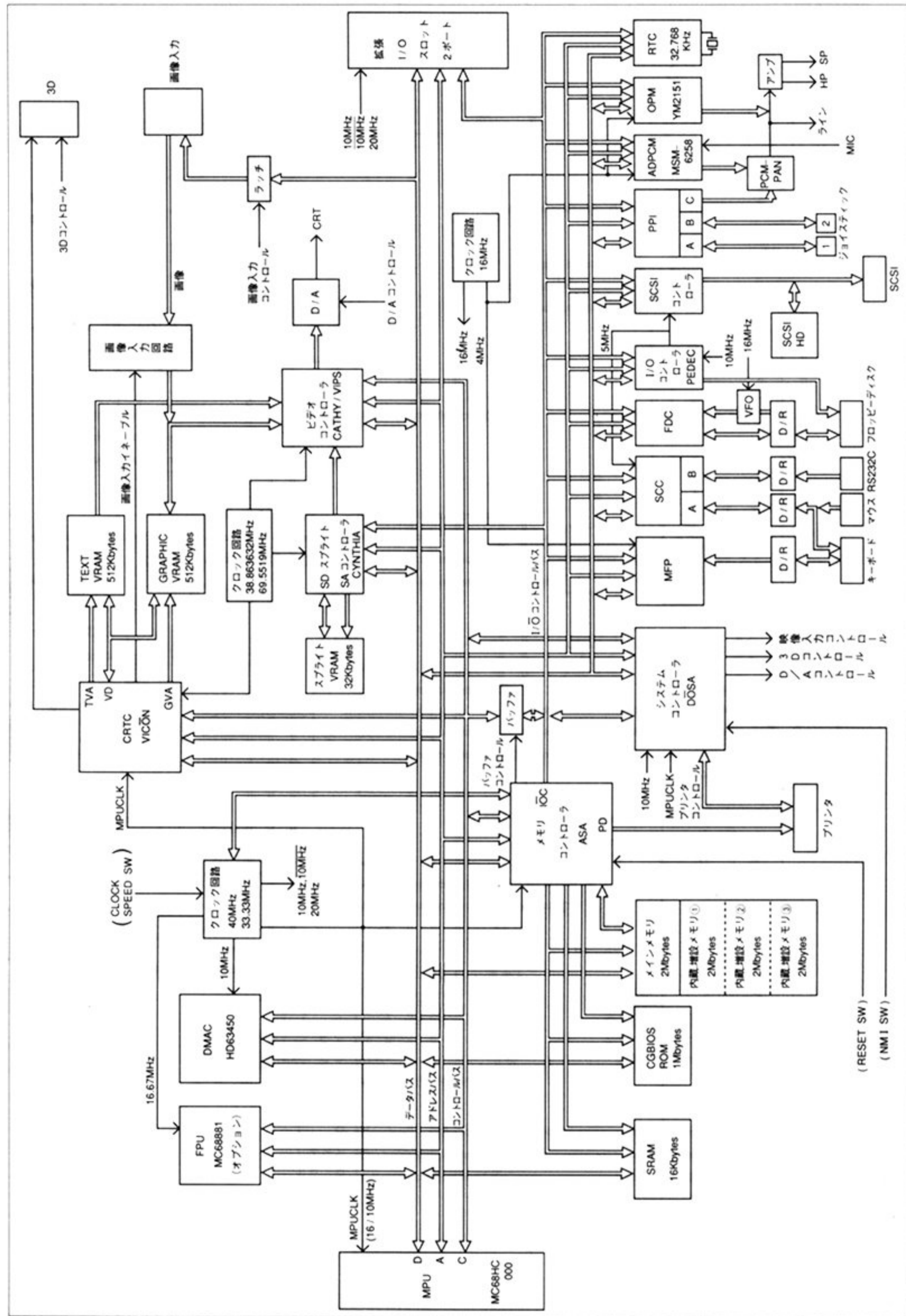
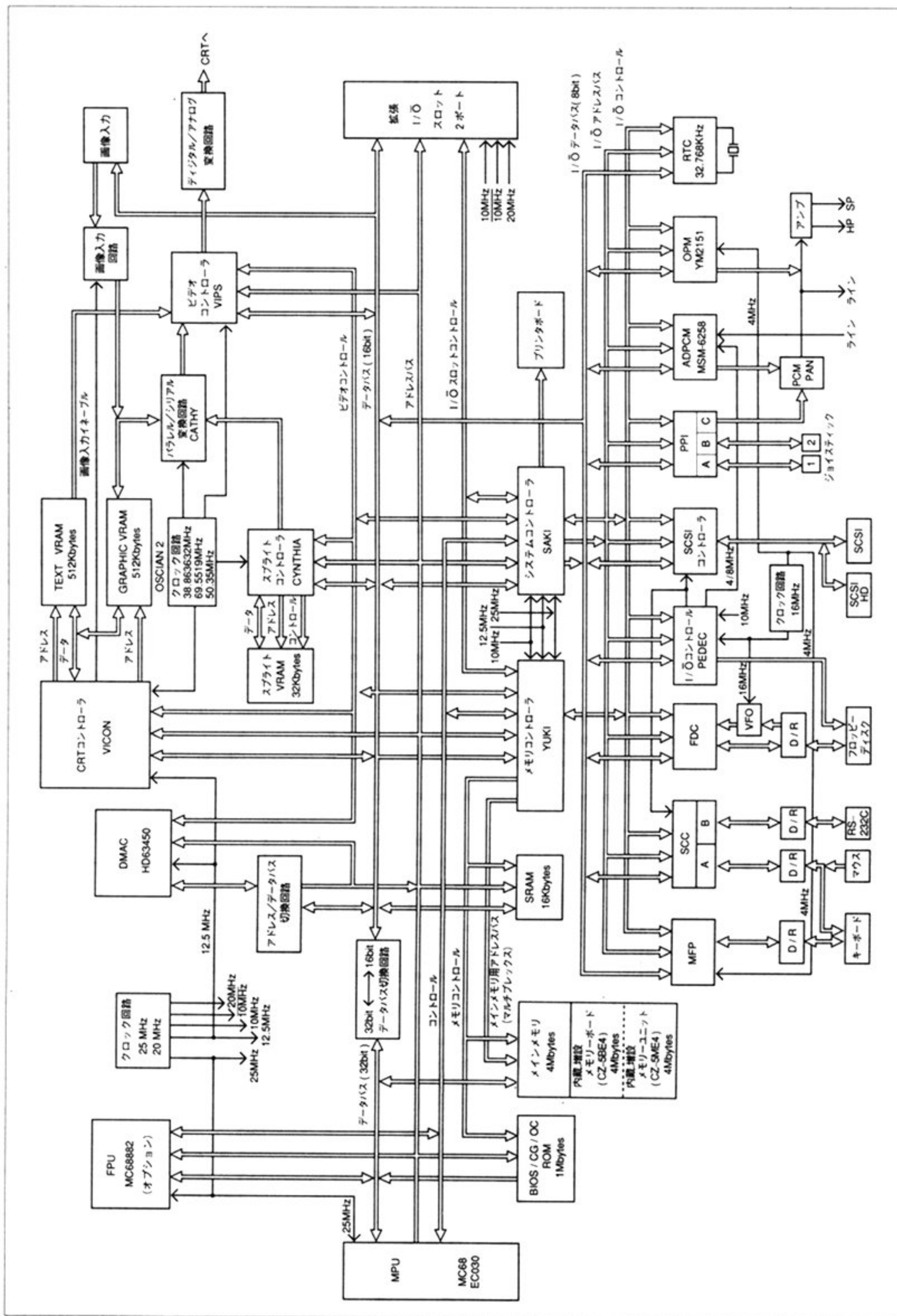
また，640×480ドットモードをサポートするため，CRTインターフェース系の基本クロック（ドットクロック）を生成するLSI（クロックジェネレータ）も変更されました。変更されたLSIの一覧を図1に示します。

●図1……X68030で変更された主要LSI

種　別	X68000XVI	X68030
メモリコントローラ	iX1748CE(ASA)	iX2136CE(YUKI)
システムコントローラ	iX1749CE(DŌSA)	iX2137CE(SAKI)
クロックジェネレータ	iX1096CE(OSCIAN)	iX1856CE(OSCIAN2)

③内部ブロックの変更

X68000XVIのブロック図を図2に，X68030のブロック図を図3に示します。X68000シリーズの内部I/Oデバイスや拡張スロットなどのアクセスタイミングは，10MHzのMC68000の動作タイミングを基本として設計されています。X68000XVIになって，CPUは16MHz，DMA

●図2……X68000XVIのブロック図

●図3……X68030のブロック図

コントローラなどのI/Oデバイスや拡張スロットは10MHzと異なるものになりました。この違いをシステムコントローラ（DŌSA）が吸収しています。

X68030では，CPUのデータバス幅や動作タイミングがまったく異なるものとなり，さらにDMACとCRTインターフェース関係が16MHz, I/Oスロットが10MHzになるなど，動作タイミングの異なるブロックが増えたためにX68000XVIよりも複雑なものとなっています。

CPUとROM，メインメモリ，数値演算プロセッサが，CPUと同じ32ビット幅のバスに接続されます。そのほかのI/Oデバイス類は16ビット幅のMC68000相当のバスであるため，システムコントローラ(SAKI)によってバス幅とタイミングの変換が行われます。システムコントローラでは，DMA/CRTインターフェース系，拡張スロット系，内部I/Oデバイス系のそれぞれのブロックごとに異なる動作タイミングを生成しています。

④数値演算プロセッサの変更

数値演算プロセッサは，X68000シリーズではMC68881，X68030ではバージョンアップ版に相当するMC68882が使用されています。MC68882はMC68881と互換性があり，差し替えて使うことも不可能ではないLSIです（ただし，X68000シリーズとX68030シリーズでは形状の異なるものを使っているため，X68030用のものをそのままX68000シリーズで使うことはできません)。

X68000シリーズとX68030シリーズの決定的な違いは，X68000シリーズでは数値演算プロセッサを通常のI/Oデバイスと同様に接続するしか方法がなかったのに対し，X68030シリーズではCPUと直結してCPU自身が持っているコプロセッサ命令で動作させることができるようになったことです。これにより前著『Inside X68000』（ソフトバンク刊）で行ったような面倒なやりとりはすべてCPU自身で行うようになり，取り扱いが格段にやさしくなったうえ，演算速度も大幅に向上しました。

⑤ROMの変更

X68000/X68030シリーズでは，ROM領域として$F00000から$FFFFFFの１Mバイトの領域が割り当てられています。X68000XVI，X68030とも，この空間は２個のマスクROMを使い，BIOSと文字フォントの両方を詰め込んでいます。

ただし，X68000シリーズでは，データバス幅が16ビットであったため，512K×８ビットの4Mビットマスク ROMを２つ使っていました。X68030では，CPUのデータバス幅が32ビットになったため，512K×16ビットの8Mビットマスク ROMを２個使っている点が異なります。ROM１個あたりの容量が増えたため，ROMの総容量は2Mバイトとなりますが，X68000/X68030シリーズではROM用の空間は1Mバイトしかとられていませんので，X68030ではROMのアドレスピンのうち最上位のピンを0Vに固定することで，256K×16ビット相当の

ROMとして使っています。

マスクROMの型名は，X68000XVIでは上位８ビットがiX1775CE，下位８ビットがiX1776CEで，X68030では上位16ビットがiX2138CE，下位16ビットがiX2139CEとなっています。

⑥メモリ増設方法の変更

X68000/X68030シリーズでは，メインメモリ空間として12Mバイト分が割り振られています。初代X68000では，内部には2Mバイトまでのメモリしか実装できず，2Mバイト以上のメモリを実装するには拡張スロットにメモリボードを入れて対応していました。X68000XVIよりも前の機種では，拡張スロットの動作とCPUの動作タイミングが同一でしたのでこれでもよかったのですが，X68000XVI以降，拡張スロットよりもCPUの動作タイミングが速くなったため，拡張スロットでメインメモリを増設するのではCPUの本来の性能が引き出せなくなりました。このため，X68000XVIでは内部増設できるメモリ容量を最大8Mバイトと大幅に増やしています。さらに，X68030では内部だけで12Mバイトまで増設できるようにしています。

メモリ増設の対応は，初代X68000などでは内部に1Mバイトのメモリボードを取り付けられるようになっているだけでしたが，X68000XVIやX68030では内部拡張用のメモリモジュールを取り付け，さらにそのボードの上に増設メモリボードを取り付けるという方法をとっています。

X68000XVIでは，本体の標準メモリが2Mバイトあり，内部増設ボード（CZ-6BE2A）で2Mバイト，さらにCZ-6BE2Aの上に乗る形で実装される2Mバイト増設メモリモジュール(CZ-6BE2B)を２枚まで取り付けることができるため，計8Mバイトまで内部実装可能となっています。

X68030では，本体の標準メモリが4Mバイトと増え，さらに内部増設ボード(CZ-5BE4)が4Mバイト，増設メモリモジュール（CZ-5ME4）が4Mバイトとなっており，計12Mバイトを内部に実装できるようになっています。

⑦SCSI IDの設定方法の変更

X68000XVIの内蔵HDDはソフトウェア的にIDを変更できる物が使われていましたが，X68030ではハードウェアによってID設定を行うものに変更されています。

X68030のHDDインターフェースの信号は，コントロール基板，HDDコネクタ基板，内蔵HDDの３つを40芯のフラットケーブルで接続しており，HDD側ではこのうち３本をID設定用の信号として見るようにしています。このため，X68030ではHDDコネクタ基板の上にロータリースイッチを設け，３本のID設定信号の状態を変えるようにしています。コントロール基板側では，ID設定信号ピンはN.C.（無接続）として無視しています。

⑧コネクタの廃止，形状／信号の変更

X68030ではコネクタ類をCompactタイプと統一しようと考えたためか，コネクタの形状変更や，マンハッタンシェイプタイプのものでも一部使用頻度の低いコネクタの廃止などが行われています。これにより従来のX68000シリーズの周辺装置が使えなくなったり，ケーブルの変更が必要になるものがあります。以下に，廃止・変更されたコネクタについてまとめておきます。

ステレオスコープ端子廃止（信号自体も廃止）

ステレオスコープは，左右の目のそれぞれに液晶シャッターが取り付けられたメガネのようなものです。シャッターは本体からの制御信号でそれぞれ独立して開閉することができますので，画面の切り替えと連動してシャッターを動かすことで，左右の目に異なる映像を与え，奥行き感を出すことができます。VR（バーチャルリアリティ）を実現するためのアダプタと考えてもよいでしょう。

ステレオスコープはX68000の発売当初，シャープからオプションとして発売されており，対応したゲームも出たのですが，その後はあまり積極的に利用されなくなりました。X68000シリーズでも，Compactタイプではステレオスコープ端子が廃止されていましたが，X68030ではマンハッタンシェイプタイプでも廃止されてしまいました。

VRは，ゲームだけでなく，平面のディスプレイだけでは表現しにくい現象を解析するときなどに使われ始め，むしろこれから必要となってくる機能であるように思えるだけに残念な気もします。

シースルーカラー端子廃止

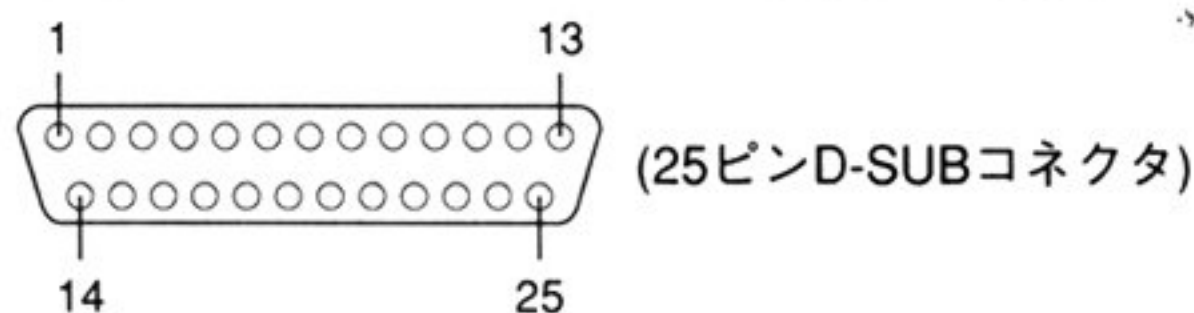
シースルーカラー端子は，カラーイメージユニットと接続してコンピュータ画面とビデオ画面の半透明合成を行わせるための信号端子です。X68030では，この信号端子が削除されました。そのかわり，イメージユニットと接続するもう1つのコネクタである，イメージ入力コネクタを変更してシースルーカラー信号を出すようにしています。

イメージ入力コネクタの変更

X68000のイメージ入力コネクタは25ピンのD-SUBコネクタでしたが，X68030ではハーフピッチの30ピンコネクタになっています。

X68000とX68030のイメージ入力端子のピン配置を図4と図5に示します。前にも触れたとおり，X68030ではイメージ入力コネクタが変更され，シースルーカラー端子の信号，ディスプレイTV制御信号や外部垂直／水平同期信号などが組み込まれました。これによってイメージユニットに関係する信号は，すべてイメージ入力コネクタに集約されることになりました。

ハーフピッチコネクタにはいくつか種類がありますが，X68030ではすべて3M(スリーエム)社のものと互換をとったもの(俗に「3Mコンパチブル」と呼ばれています)に統一されています。

●図4……X68000のイメージ入力端子ピン配置

番号	名 称	内　容	番号	名 称	内　容
1	ADD11	画像データ(下位12ビット)	14	VCC1	+5V
2	ADD10		15	GND	グランド
3	ADD9		16	VCC3	+12V
4	ADD8		17	CD4	イメージユニット制御信号
5	ADD7		18	CD3	
6	ADD6		19	CD2	
7	ADD5		20	CD1	
8	ADD4		21	CD0	
9	ADD3		22	ADD15	画像データ(上位4ビット)
10	ADD2		23	ADD14	
11	ADD1		24	ADD13	
12	ADD0		25	ADD12	
13	QA	ドットクロック			

●図5……X68030イメージ入力端子ピン配置

番号	名 称	内　容	番号	名 称	内　容
1	VHT	シースルーカラー	16	TVREMOTE	ディスプレイTV制御信号
2	GND	グランド	17	TV ON/OFF	ディスプレイTV ON/OFF信号
3	ADD11	画像データ(下位12ビット)	18	EX VSYNC	外部垂直同期信号
4	ADD10		19	EX HSYNC	外部水平同期信号
5	ADD9		20	VCC1	+5V
6	ADD8		21	GND	グランド
7	ADD7		22	CD4	イメージユニット制御信号
8	ADD6		23	CD3	
9	ADD5		24	CD2	
10	ADD4		25	CD1	
11	ADD3		26	CD0	
12	ADD2		27	ADD15	画像データ(上位4ビット)
13	ADD1		28	ADD14	
14	ADD0		29	ADD13	
15	QA	ドットクロック	30	ADD12	

SCSIコネクタのハーフピッチ化

X68000シリーズの拡張HDDコネクタは，マンハッタンシェイプタイプのものでは50ピンのアンフェノールコネクタ，Compact XVIではハーフピッチコネクタと異なっていましたが，

X68030ではハーフピッチコネクタに統一されました。

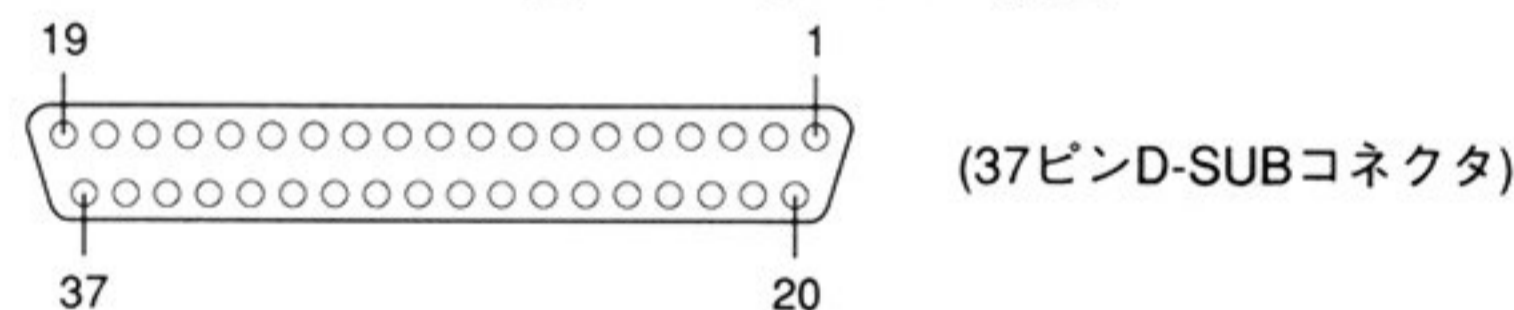
SCSIを規格化したANSIでは，当初ハーフピッチコネクタはAMP社のピンフォークタイプとしていました。このため，ワークステーションなどではピンフォークタイプを使用しているものがほとんどです。ピンフォークタイプは，D-SUB 1.27mmピッチで細いピンが伸びているものです。D-SUBコネクタを縮小したようなものだと思えばわかりやすいでしょう。あまりにピッチがつまっており，しかもピンが細いため，ピンが折れ曲がってしまうといったトラブルが少なくないようです。

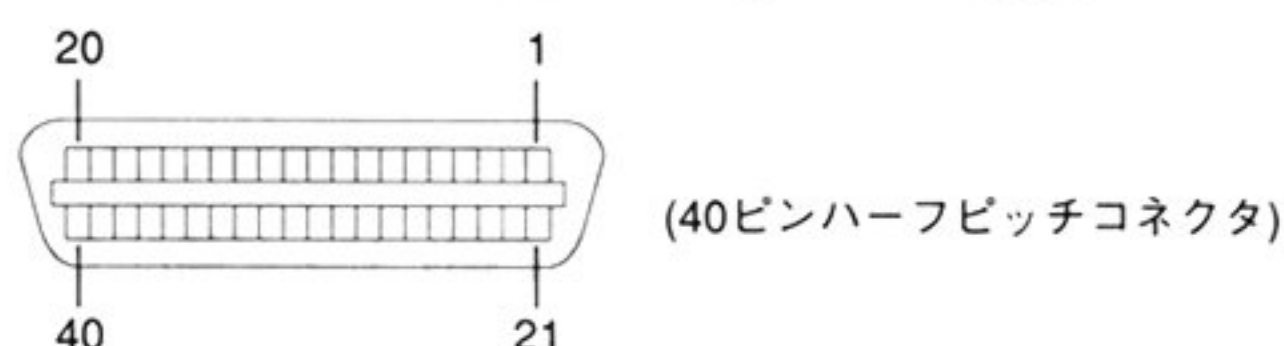
国内ではANSIの影響が弱かったためか，あるいはあらかじめこのトラブルを避けたためか，3Mコンパチブルタイプのものが主流となっています。ANSIでもピンフォークの問題を無視できなかったようで，最近3Mコンパチブルタイプのものも規格に取り入れられました。

この動きを受けたのでしょう。X68030でもSCSIコネクタは3Mコンパチブルタイプのハーフピッチコネクタとなっています。

外部FDDコネクタのハーフピッチ化

外部FDDコネクタは，X68000では37ピンのD-SUBコネクタでしたが，X68030では40ピンのハーフピッチコネクタに変更されています。X68000の外部FDDコネクタの信号配置を図6に，X68030のものを図7に示します。

●図6……X68000の外部FDDコネクタピン配置

番号	名　称	内　容	番号	名　称	内　容
1	DISK TYPE SELECT	ディスクタイプ選択	20	OPTION SELECT 0	拡張機能選択信号
2	N.C.	非接続	21	OPTION SELECT 1	
3	DRIVE SELECT 3	ドライブ#3選択	22	OPTION SELECT 2	
4	INDEX	インデックス信号	23	OPTION SELECT 3	
5	DRIVE SELECT 0	ドライブ#0～#2選択信号	24	EJECT	FDイジェクト信号
6	DRIVE SELECT 1		25	EJECT MASK	イジェクト動作マスク信号
7	DRIVE SELECT 2		26	LED BLINK	LED点滅制御信号
8	MOTOR ON	スピンドルモータON/OFF制御	27	DISK IN	ディスク挿入ステータス信号
9	DIRECTION	ヘッド移動方向設定	28	ERR DISK	ディスク誤挿入ステータス信号
10	STEP	ヘッド移動パルス信号	29	FDD INT	ディスク割り込み信号
11	WRITE DATA	書き込みデータ信号	30	GND	グランド
12	WRITE GATE	書き込みゲート信号	31	GND	
13	TRACK 00	0トラック検出信号	32	GND	
14	WRITE PROTECT	書き込み禁止ステータス	33	GND	
15	READ DATA	読み出しデータ信号	34	GND	
16	SIDE SELECT	ヘッド切り替え信号	35	GND	
17	READY	ドライブレディ信号	36	GND	
18	N.C.	非接続	37	N.C.	非接続
19	N.C				

●図7……X68030の外部FDDコネクタピン配置

番号	名　称	内　　容	番号	名　称	内　　容
1	DISK TYPE SELECT	ディスクタイプ選択	21	OPTION SELECT 1	拡張機能選択信号
2	N.C.	非接続	22	OPTION SELECT 2	
3	DRIVE SELECT 3	ドライブ#3選択	23	OPTION SELECT 3	
4	INDEX	インデックス信号	24	EJECT	FDイジェクト信号
5	DRIVE SELECT 0	ドライブ#0〜#2選択信号	25	EJECT MASK	イジェクト動作マスク信号
6	DRIVE SELECT 1		26	LED BLINK	LED点滅制御信号
7	DRIVE SELECT 2		27	DISK IN	ディスク挿入ステータス信号
8	MOTOR ON	スピンドルモータON/OFF制御	28	ERR DISK	ディスク誤挿入ステータス信号
9	DIRECTION	ヘッド移動方向設定	29	FDD INT	ディスク割り込み信号
10	STEP	ヘッド移動パルス信号	30	GND	グランド
11	WRITE DATA	書き込みデータ信号	31	GND	
12	WRITE GATE	書き込みゲート信号	32	GND	
13	TRACK00	0トラック検出信号	33	GND	
14	WRITE PROTECT	書き込み禁止ステータス	34	GND	
15	READ DATA	読み出しデータ信号	35	GND	
16	SIDE SELECT	ヘッド切り替え信号	36	GND	
17	READY	ドライブレディ信号	37	N.C.	非接続
18	N.C.	非接続	38	GND	グランド
19	N.C.		39	GND	
20	OPTION SELECT 0	拡張機能選択信号	40	GND	

⑨拡張スロット信号や動作タイミングの一部変更

X68030の拡張スロットは，X68000シリーズのものとある程度互換性を保ったものとなっていますが，あまり利用されることのない信号やDRAM制御関係の信号を省略するなどの変更が行われています。

2 CPUの違い

冒頭でも述べましたが，X68000/X68030シリーズにはモトローラのM68000ファミリーのCPUが使用されています。M68000ファミリーのCPUは，MC68000から始まり，MC68000の改良版ともいうべきMC68010，MMU（メモリマネージメントユニット）や数値演算プロセッサなどを外付けでサポートできるようにしたMC68020，MMUとキャッシュメモリを内蔵し

たMC68030，さらに数値演算プロセッサも内蔵させたMC68040が本流としてあり，これらのCPUから派生した品種が数多く存在します。

これらのCPUファミリーのうち，X68000シリーズには最も古いアーキテクチャのMC68000の低消費電力版であるMC68HC000が，X68030にはMC68030からMMU機能を取り去ったMC68EC030が使われています。これらのCPUがソフト的に完全な互換性を維持しつつ機能追加を行っているのであれば問題は少なかったのですが，残念ながら，世代が進むごとに互換性のない部分が生まれてきてしまっています。

非互換部分の多くは，一般的なアプリケーションソフトウェアなどには影響のない部分ですので，MC68000用に書かれたプログラムの多くはMC68030やMC68040でも動作します。問題があるのは，CPUの動作のうち，かなりクリティカルな部分を操作するソフトウェアです。特にOSのように，ハードウェアのアーキテクチャに依存する部分の多いソフトウェアではこの違いは致命的で，それぞれのCPUにあわせて書き換える必要があります。

ゲームでも，最近のものは高速化のためにCPUの動作を逆手にとるようなトリッキーなプログラミングをしたものがありますが，このようなものはCPUが変わると動かなくなる可能性が非常に高いといえます。

メモリや周辺LSIのアドレスがまったく同じであるにもかかわらず，X68030で動かないX68000用ソフトウェアがあるというのは，CPUの動作が速くなったことよりも，むしろCPUの動作の非互換部分の影響によるものがほとんどです。

現在，X68000シリーズで気をつけなくてはならないのは，MC68000とMC68EC030の違いのみですが，今後，X68000/X68030シリーズにも，MC68040を搭載した上位機種や，MC68EC030のかわりにMMU機能のあるMC68030が搭載される可能性もないとはいえません。ここでは，MC68000からMC68040に至るM68000ファミリーのCPUの主な違いをまとめておくことにします。

2・1 M68000ファミリーCPUの比較

M68000ファミリーのCPUとしては，MC68000/68010/68020/68030/68040の５種類が主流として存在します。これら５種類のCPUの比較表を図８に示します。

MC68010は，MC68000でサポートしかけていた仮想記憶サポート機構のバグを修正したうえで一部機能強化を図ったようなもので，大きな変更は行われていません。

MC68020は，MC68000のアーキテクチャをベースに本格的な32ビットCPUとして発展させたものです。M68000ファミリーCPUのアーキテクチャは最初から32ビットCPUでしたが，外部のデータバスも32ビットとなって，真に32ビットCPUとなったのはMC68020以降です。

●図8……M68000ファミリーのCPUの比較

<table>
<tr><th></th><th>MC68000</th><th>MC68010</th><th>MC68020</th><th>MC68030</th><th>MC68040</th></tr>
<tr><td>データバスサイズ</td><td>16</td><td>16</td><td>8/16/32</td><td>8/16/32</td><td>32</td></tr>
<tr><td>アドレスバスサイズ</td><td>24</td><td>24</td><td>32</td><td>32</td><td>32</td></tr>
<tr><td>命令キャッシュ</td><td>なし</td><td>3ワード</td><td>256バイト</td><td>256バイト</td><td>4Kバイト</td></tr>
<tr><td>データキャッシュ</td><td colspan="3">なし</td><td>256バイト</td><td>4Kバイト</td></tr>
<tr><td>MMU</td><td>対応不可</td><td colspan="2">外付けMMU(MC68851)対応</td><td colspan="2">内蔵</td></tr>
<tr><td>数値演算プロセッサ</td><td colspan="2">I/Oデバイスとして接続</td><td colspan="2">コプロセッサとして接続</td><td>内蔵</td></tr>
<tr><td>アライメントの制約</td><td colspan="2">ワード/ロングワード/データ，命令，スタックはワード境界に整列しなくてはならない</td><td colspan="3">命令はワード境界に整列しなくてはならない（データ，スタックは整列させなくてもよい）</td></tr>
</table>

MC68020は，以降のM68000ファミリーCPUの基礎となるアーキテクチャを確立したCPUであるといえます。

MC68030はMC68020用の外付けMMUのサブセット版を内蔵させたもの，MC68040はMC68030のMMU機能をさらに縮小したうえで数値演算プロセッサ（MC68020用の数値演算プロセッサMC68882のサブセット版）を内蔵させたものです。

2・2 MC68020以降で追加／削除された命令

M68000ファミリーのCPUがMC68020で大きく変更されたことはすでに述べました。MC68020以降で追加された命令を図９に示します。これらの命令は，MC68000およびMC68010にはありません。

図中，MC68020/MC68030に浮動小数点演算命令が，MC68020にMMU関係の命令がないことになっているのは，CPUに浮動小数点プロセッサやMMUの機能が内蔵されていないためで，外付けのコプロセッサ（MC68881/MC68882，MC68851）を取り付ければ利用できるようになります。これら以外の変更点を整理すると，次のようになります。

●図9……MC68020以降の追加命令

命令	注記	MC68020	MC68030	MC68040
Bcc	32ビットディスプレースメントをサポート	○	○	○
BFxxx	ビットフィールド (BFCHG,BFCLR,BFEXTS,BFEXTU BFFO,BFINS,BFSET,BFTST)	○	○	○
BKPT	(ブレークポイント命令)	○	○	×
BRA	32ビットディスプレースメントをサポート	○	○	○
BSR	32ビットディスプレースメントをサポート	○	○	○
CALLM	(モジュール間呼び出し命令)	○	×	×
CAS/CAS2	(比較／交換命令)	○	○	○
CHK	32ビットオペランドをサポート	○	○	○
CHK2	(上下限チェック命令)	○	○	○
CINV	(キャッシュメインテナンス命令)	×	×	○
CMPI	PC相対アドレッシングモードをサポート	○	○	○
CMP2	(レジスタの上下限比較命令)	○	○	○
CPUSH	(キャッシュメインテナンス命令)	×	×	○
cp	(外付けコプロセッサ命令)	○	○	×
DIVS/DIVU	32ビット／64ビット操作をサポート	○	○	○
LINK	32ビットディスプレースメントをサポート	○	○	○
MOVE16	16バイトブロック転送命令	×	×	○
MOVEC	追加された制御レジスタへの転送	○	○	○
MULS/MULU	32ビット／64ビットの結果をサポート	○	○	○
PACK	(パック形式BCD)	○	○	○
RTM	(モジュール間復帰命令)	○	×	×
TST	PC相対アドレッシングをサポート	○	○	○
TRAPcc	条件付きトラップ命令	○	○	○
UNPK	(アンパックBCD)	○	○	○

2·2·1 MC68030以降で省略された命令

MC68020で実装されながらMC68030以降で省略された命令には，CALLM命令とRTM命令があります。これは，タスクのレベル管理機能を使ったときに利用される命令です。MC68020は，MMU (MC68851) を取り付けると8段階の特権レベル管理が行えるようになり，下位のレベルで動いているときに上位のレベル付けがされているメモリ領域を操作することを禁止するなどのメモリ保護機能を使うことができるようになります。

このとき，下位レベルのプログラムが上位のプログラムを呼び出す(アプリケーションがOSのサービスルーチンを利用するような場合) ときに使われるのがCALLM命令，逆に上位から下位に戻るのに使われるのがRTM命令です。

MC68030以降のCPUではMMU機能が内蔵されましたが，いずれも若干の機能的な追加は

あるものの，基本的にはMC68851のサブセット版となっており，レベル管理機構は省略されています。このため，CPUからもCALLM/RTM命令が省略されたというわけです。

2・2・2 MC68040で省略された命令

MC68040では，BKPT命令や外付けのコプロセッサ用の命令が省略されています。BKPT（ブレークポイント）命令は，CPUの外部バス上でブレークポイントアクノリッジバスサイクルを実行するものです。CPUがブレークポイントアクノリッジサイクルを実行したとき，外部回路からデータを返すと，CPUはそれを命令として取り込みます。これによって，通常の命令処理の途中に任意の命令を挟み込ませることができるわけです。

この機能は，外部回路のサポートが必須であることのほか，ソフトによる命令コードの置き換えなどによるブレークポイントで十分であったこともあってか，MC68040では削除されています。

また，MC68040では外付けのコプロセッサ用の制御命令が削除されています。MC68020で用意された外付けのコプロセッサには数値演算プロセッサとMMUがありました。MC68020以降に実装されたコプロセッサインターフェースは汎用性の高いもので，これら以外のものにも利用できる可能性があったのですが，結局，新しいコプロセッサは開発されることはありませんでした。

MC68040では，（サブセット版ですが）この両方が内蔵されたため，外付けのコプロセッサ用の命令はサポートされなくなりました。コプロセッサを外付けした場合には，X68000のときのようにソフトウェアでコプロセッサを制御する必要があります。

2・2・3 MC68040で追加された命令

MC68040の内蔵キャッシュメモリは，MC68020/MC68030がそれぞれ256バイト/512バイトであったのに対し，8Kバイトと大幅に増加しています。このため，MC68040ではキャッシュメモリ制御のための命令が２つ追加されています。CINVは選択したキャッシュラインを無効化する命令，CPUSHは選択したダーティキャッシュラインの内容を退避した後，キャッシュラインを無効とするものです。

MC68030では，キャッシュ充填時のバス動作の制御やクリア，イネーブル／ディセーブルなどの制御をCACR（キャッシュコントロールレジスタ）で行っていましたが，MC68040ではCINV，CPUSH命令の追加にともない，CACRは単にキャッシュのイネーブル／ディセーブ

ルを行うだけのものになっています。

3 エリアセット

エリアセット機能は，MMUを持たないX68000/X68030シリーズに採用された唯一のメモリ保護機構です。X68030は，MC68030からMMU機能を外したMC68EC030を使用したためか，エリアセット機能の強化が図られています。ここでエリアセット機能の変更点について説明することにします。

3・1 エリアセット機能

MC68000ファミリーのCPUは，スーパーバイザモードとユーザモードの２つの動作モードを持っています。スーパーバイザモードはOSなどのシステムプログラム，ユーザモードはアプリケーションというように区別して動作させることを考慮したものです。スーパーバイザモードのときにはCPUの持つ命令をすべて使うことができますが，ユーザモードではシステムに影響を与えるような一部の命令が使用できなくなっています。

これに加え，CPUには外部バスのアクセスを行うときに，そのアクセスがスーパーバイザモードで行われるものか，ユーザモードで行われるものか，また，データアクセスなのか，プログラムの読み出しなのかといった情報を示す端子があります。外部回路でピンの状態をチェックすることで，ある領域をスーパーバイザ領域（スーパーバイザモードでなければアクセスできない領域）とすることができるようになります。これによって，ユーザモードのプログラムがOSの領域を破壊するなどの問題を避けることができ，システムとしての安全性を高めることができるわけです。

X68000では，このCPUの機能を利用し，ベクタテーブルなどがある$000000番地以降，どれだけの空間をスーパーバイザ領域とするかを指定することができるようになっています。また，X68030ではこれに加えて2Mバイトから12Mバイトまでの空間（$200000～$BFFFFF）を40個のブロックに分割し（１ブロックあたり256Kバイトになります），個々のブロックごとにスーパーバイザ領域とするか否かを設定できるようにしています。

エリアセットのレジスタの一覧を図10に示します。X68000から持っていたエリアセットレジスタは１つ，X68030から追加された拡張エリアセットレジスタは５つあります。

●図10……エリアセットレジスタ／拡張エリアセットレジスタアドレス配置

レジスタ種別		レジスタアドレス	管理する領域
エリアセットレジスタ		$E86001	$000000～$1FFFFF
拡張エリアセットレジスタ	0	$EAFF81	$200000～$3FFFFF
	1	$EAFF83	$400000～$5FFFFF
	2	$EAFF85	$600000～$7FFFFF
	3	$EAFF87	$800000～$9FFFFF
	4	$EAFF89	$A00000～$BFFFFF

3・2 エリアセットレジスタ

X68000から搭載されている，スーパーバイザ領域指定用のレジスタです。ビット配置は図11のようになっています。エリアセットレジスタは8ビット長のレジスタで，0番地からどれだけのメモリ領域をスーパーバイザ領域とするかを8Kバイト単位で指定します。値が0のときには0番地から$1FFF番地までの8Kバイト，1のときには16Kバイト……となっており，$FFを設定すると$1FFFFFまでの2Mバイトがスーパーバイザ領域となります。

X68000では，スーパーバイザ領域より上位のメインメモリ領域はユーザモード／スーパーバイザモードのいずれであってもアクセス可能です。X68030の場合，2Mバイト以上の空間には，次に述べる拡張エリアセットレジスタが働きます。

●図11……エリアセットポート(X68000/X68030共通)

スーパーバイザ領域(スーパーバイザモードでのみアクセスできるメモリ領域)の上限を8Kバイト単位で設定する。
$000000～((設定値+1)×$2000−1)までがスーパーバイザ領域になる。

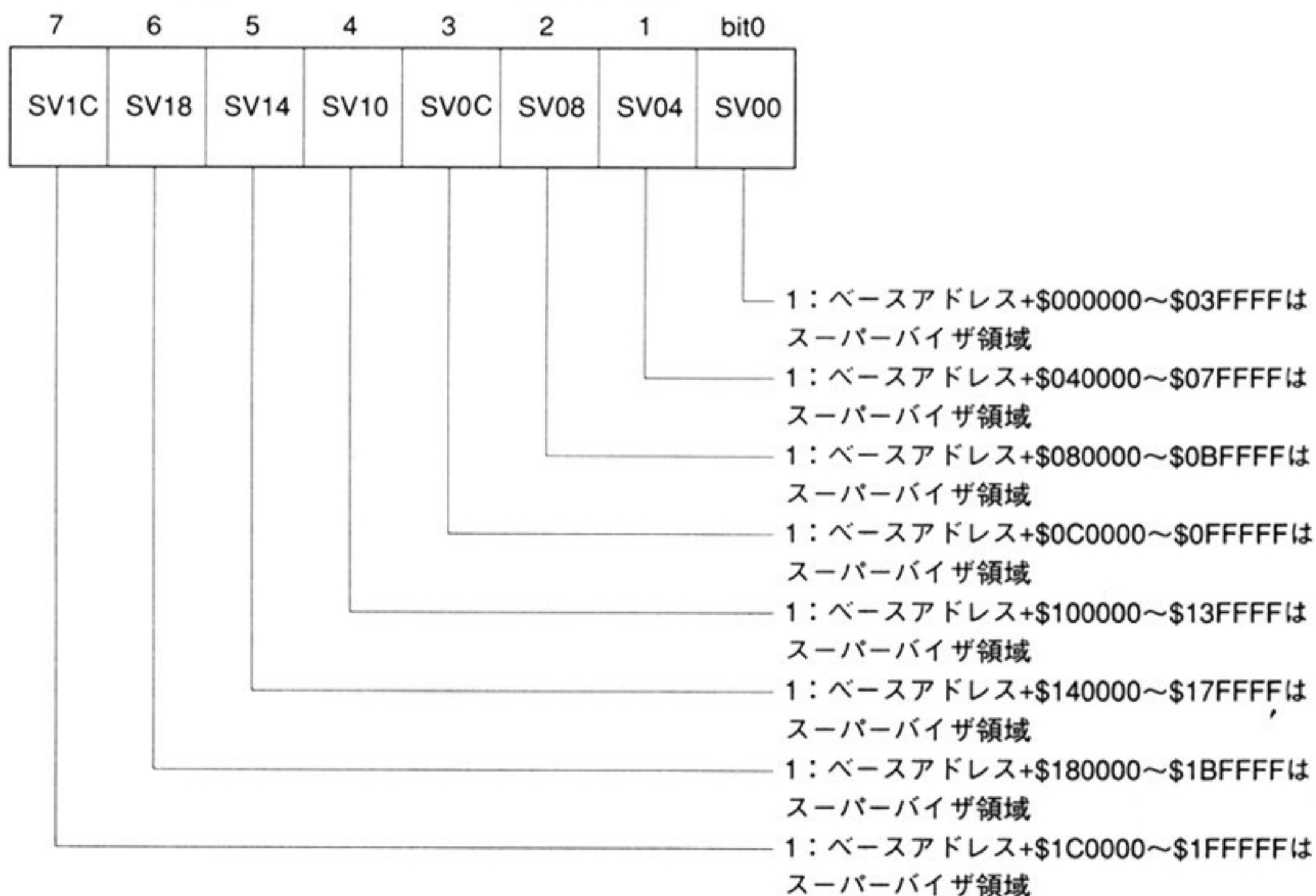
$00:0～$001FFF
$01:0～$003FFF
$02:0～$005FFF
⋮ ⋮
$7F:0～$0FFFFF (1Mバイト)
$80:0～$101FFF
⋮ ⋮
$FE:0～$1FDFFF
$FF:0～$1FFFFF (2Mバイト)

3・3 拡張エリアセットレジスタ

X68030から追加されたスーパーバイザ領域指定用のレジスタで，$200000（2Mバイト）以上のメモリ領域に対してスーパーバイザ領域の設定を行うものです。ビット配置を図12に示します。拡張エリアセットレジスタは８ビット長のものが５本用意されています。

エリアセットレジスタがスーパーバイザ領域のサイズを指定するものであったのに対して拡張エリアセットレジスタは，レジスタの各ビットがそれぞれメモリ領域を256Kバイトずつに分割したブロックに対応しており，個別にスーパーバイザ領域とするか否かを設定できるようになっている点が異なります。

$200000（2Mバイト）から$BFFFFF（12Mバイト）までのメモリに対応するため，拡張エリアセットレジスタは５本（40ビット分）用意されています。拡張エリアセットレジスタ#0の最下位ビットが$200000～$23FFFF，ビット１が$240000～$27FFFF……と順に対応し，拡張エリアセットレジスタ#4の最上位ビットが$BC0000～$BFFFFFのブロックに対応しています。対応するビットが'1'になっていると，その領域がスーパーバイザ領域になり，'0'になっていると，ユーザモード／スーパーバイザモードのいずれであってもアクセス可能な領域に

●図12……拡張エリアセットレジスタ(X68030)

なります。

リセット後，拡張エリアセットレジスタはすべて$00になっていますので，$200000以上のメモリ領域はすべてユーザモードでアクセス可能となっています。

4 数値演算プロセッサ

MC68020/68030ではCPUにコプロセッサインターフェース機能が設けられたため，数値演算プロセッサは，あたかもCPU自身の機能が拡張されたかのようにごく自然に扱うことができ

●図13……MC68040で削除された浮動小数点命令

命令	演算内容
FACOS	アークコサイン
FASIN	アークサイン
FATAN	アークタンジェント
FCOS	コサイン
FCOSH	双曲コサイン
FETOX	e^x
FETOXL	e^x-1
FGETEXP	指数ゲット
FGETMAN	仮数ゲット
FINT	整数部
FINTRZ	整数部(0方向へ丸め)
FLOG10	Log_{10} X
FLOG2	Log_2 X
FLOGN	Log_e X
FLOGNP1	Log_e (X+1)
FSQRT	平方根
FMOD	モジュロ剰余
FMOVECR	定数ROM読み出し
FREM	IEEE剰余
FSCALE	スケール指数
FSGLDIV	単精度除算
FSFLMUL	単精度乗算
FSIN	サイン
FSINCOS	サイン&コサイン
SINH	双曲サイン
FTAN	タンジェント
FTANH	双曲タンジェント
FTENTOX	10^x
FTWOTOX	2^x

るようになっています。これに対しMC68000/68010では，数値演算プロセッサは通常のI/Oデバイスと同様に接続するしか方法がありませんでした。

MC68040は，それまで外付けであった数値演算プロセッサを内蔵しましたが，機能的には外付けのものよりもかなり簡略化されたものになっています。M68000ファミリーCPU用の外付け数値演算プロセッサMC68881とその改良版であるMC68882の詳細は『Inside X68000』と，本書の数値演算プロセッサの章で説明していますので，参照してください。

MC68040で削除された浮動小数点命令の一覧を図13に示します。これらの命令を実行すると，未実装浮動小数点命令例外となります。この例外ベクタとしてはFライン例外が利用されます。

Fライン例外の処理プログラムでは，これらの命令をソフトウェアでエミュレーションすることになります。この作業を助けるため，CPUは命令の一部をデコードし，Fライン例外を発生する前にオペランドのフェッチなどを行います。

未実装浮動小数点命令例外が発生した場合，スタックに積まれているプログラムカウンタの値は，例外を発生した次の命令を指しています。例外を発生した浮動小数点演算命令のアドレスはFPIARに保持されています。

5 キャッシュメモリ

キャッシュメモリは，メインメモリの一部の内容を保持しておく高速メモリです。大容量のメインメモリをローコストに実現するためには，メモリのアクセス時間はほどほどのもので我慢しなければなりません。ところが，このままではCPUがいくら高速化されても，メインメモリの遅さに引きずられて本来の性能を引き出すことができないことになります。この解決策として生まれたのが，小容量の高速メモリを設けておいて頻繁にアクセスされるメモリ領域のコピーを保持させ，CPUからのアクセスがあったときにはこの高速メモリのデータを引き渡す方法です。この目的で使用される小容量の高速メモリを「キャッシュメモリ」と呼びます。

M68000ファミリーのCPUでは，MC68020以降CPU内部に本格的なキャッシュメモリが内蔵されるようになりました。MC68020には命令コード用のキャッシュメモリが256バイト，MC68030ではコードとデータ用にそれぞれ256バイトずつ(計512バイト)，MC68040では大幅に増えてそれぞれ4096バイト（計8Kバイト）のキャッシュメモリが内蔵されています。

図では，MC68010にも３ワード（６バイト）のキャッシュメモリがあるということにしてありますが，このキャッシュは一般的なキャッシュメモリとは性格が異なり，３ワードで収まる

小さいループ命令，たとえば，

```
lop:
move.w  (a0)+,(a2)+
dbra    d0,lop:
```

のような命令処理の高速化のために設けられたバッファメモリです。容量があまりに少ないことと，ループ命令の実行時にしか働かないものなので，モトローラはキャッシュとは呼ばず，単にこのときの動作を「ループモード」と呼ぶにとどめています。

このバッファはMC68010の特徴の１つなのですが，表には入れる項目がないため，格上げしてキャッシュの項に入れてあります。

5・1 キャッシュメモリの考え方

キャッシュメモリを持ったシステムの場合，最初のアクセスはメインメモリから読み出され，CPUにそのデータが渡されるのと同時にキャッシュメモリにも同じデータが取り込まれます。2回目以降同じ番地を読み出すと，メインメモリは読み出されず，キャッシュメモリの内容を読むことになるため，高速に動作できることになります。

このように，キャッシュメモリは２回目以降のアクセスを高速化するという特徴がありますが，キャッシュメモリの容量は有限であるため，アクセスしたすべての番地のデータを記憶しておくことはできません。なるべく頻繁にアクセスされるアドレスを記憶しておくのがよいのですが，ハードウェア的に実現するのは難しいため，一般的にはアクセスされたものを順に記憶しておき，キャッシュメモリがいっぱいになっていれば最も古いものを捨てて新しいものと入れ替えるという方法がよく使われます。

5・2 キャッシュのヒット率

CPUがアクセスした番地の内容がキャッシュメモリに入っていなかった場合を「キャッシュミス」，キャッシュメモリに入っていた場合を「キャッシュヒット」，キャッシュがヒットする割合を「キャッシュのヒット率」と呼びます。

一般的に，キャッシュのヒット率はキャッシュメモリの容量が大きくなるほど高くなりますが，ヒット率の増加の度合いは鈍っていきます。一般的なアプリケーションでは大きくヒット

率が伸びるのは4～8Kバイト程度までで，それ以降はやや鈍るといわれています。

また，キャッシュメモリに使われるような高速動作が可能なメモリチップは非常に高価です。CPUに内蔵させるときには，キャッシュメモリの容量増加はそのままCPUのチップ面積の増大に反映し，CPUの価格上昇につながります。いずれにせよ，キャッシュメモリの増加は，そのまま製品価格にはねかえります。キャッシュメモリを搭載する場合，性能向上とコスト増加のバランスをどのあたりでとるかというのはなかなか難しい問題です。

5・3 キャッシュの方式

キャッシュメモリは小容量のメモリを大きなメインメモリの一部のコピーとして使うため，データを格納する部分とは別に，そのデータがメインメモリのどの番地の内容であるかなどの付帯情報が必要になります。この情報を「タグ」と呼びます。キャッシュメモリは，このタグ情報とCPUからアクセスしようとしているアドレス情報を比較し，一致するものがあれば，そのタグと対応して格納されているデータを引き渡すわけです。アドレスの比較やデータの取り出しは特に高速な動作が要求されるため，すべてハードウェアで実現されます。

キャッシュメモリの実現方法としては次のようなものが考えられています。

・ダイレクトマップ方式

・フルアソシエイティブ方式

・セットアソシエイティブ方式

・セクタ方式

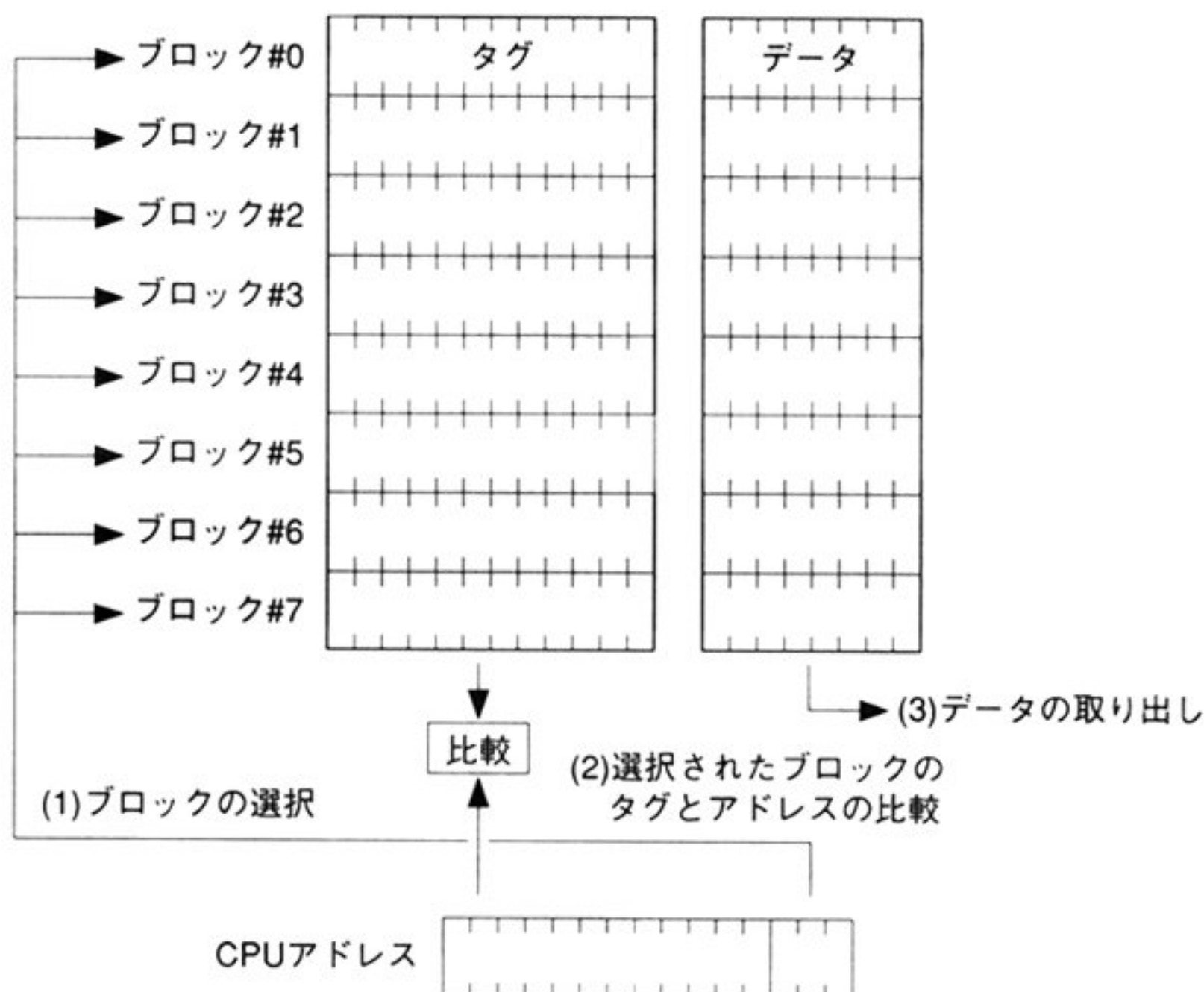
これらのうち，マイクロプロセッサでよく利用されているのは，ダイレクトマップ方式とセットアソシエイティブ方式の2つです。ダイレクトマップ方式は比較的小容量のものを利用するときに使われます。逆に，数Kバイト以上の容量になるとセットアソシエイティブ方式が使われることが多くなっています。

M68000ファミリーCPUでは，MC68020とMC68030がダイレクトマップ方式，MC68040ではセットアソシエイティブ（4ウェイセットアソシエイティブ）方式が採用されています。

次に，これらの方法がどのようなものであるかを説明しておきましょう。

5・3・1 ダイレクトマップ方式

ダイレクトマップ方式は，アドレス情報の一部をそのままキャッシュメモリの選択に利用するものです。図14にダイレクトマップ方式でキャッシュメモリを実現した例を示します。説明

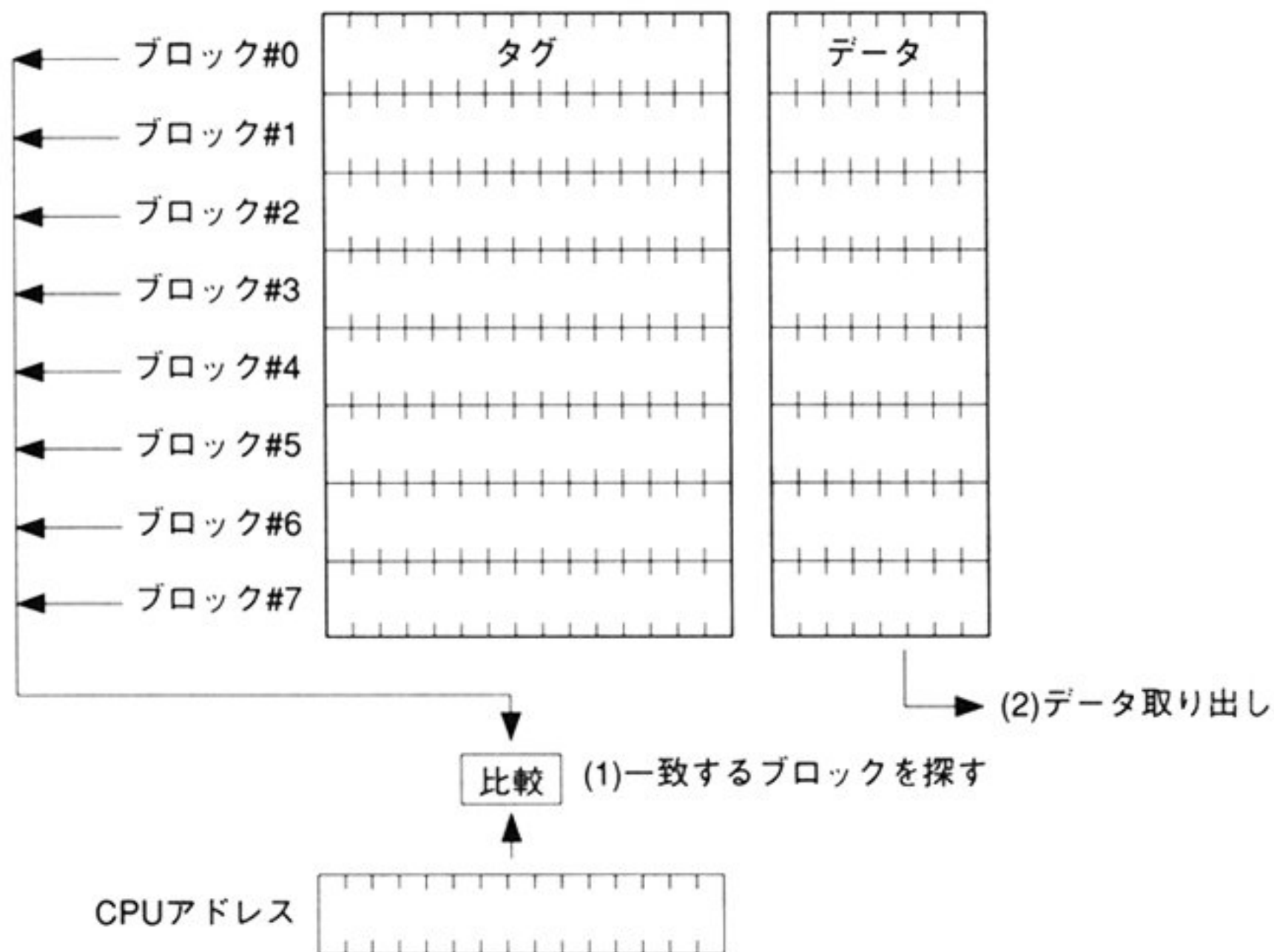
●図14……ダイレクトマップ方式

を簡略化するため，この例では1ブロック（キャッシュメモリに格納されるデータの単位）は1バイトになっており，全部で8ブロックあることにしています。このとき，キャッシュメモリの容量は8バイトということになります。一方，CPUが出力するアドレスは16本であるとしましょう。

タグ情報は各ブロックごとに持っています。ダイレクトマップ方式では，このCPUが出力したアドレスのうち3ビット分をブロックの選択に使い，残り13本のアドレスをキャッシュのタグ情報と比較します。タグ情報とアドレス情報の13ビットのデータが一致すればキャッシュヒットとなり，不一致であればキャッシュミスとなります。

通常，CPUによるメモリアクセスは狭い範囲に集中しますので，ブロックの選択に使うビットはアドレスの下位のほうからとるのが普通です。この例でもブロック選択はアドレスの下位3ビットで行っています。

5・3・2 フルアソシエイティブ方式

ダイレクトマップ方式は，ヒット／ミスの判定方法が簡単で回路も小規模なものですみますが，CPUが不連続なメモリ領域をアクセスしたような場合，キャッシュ選択用のビットが一致して，タグが一致しない場合が増えてきます。これでは，キャッシュメモリが有効に使われて

●図15……フルアソシエイティブ方式

いるとはいえません。

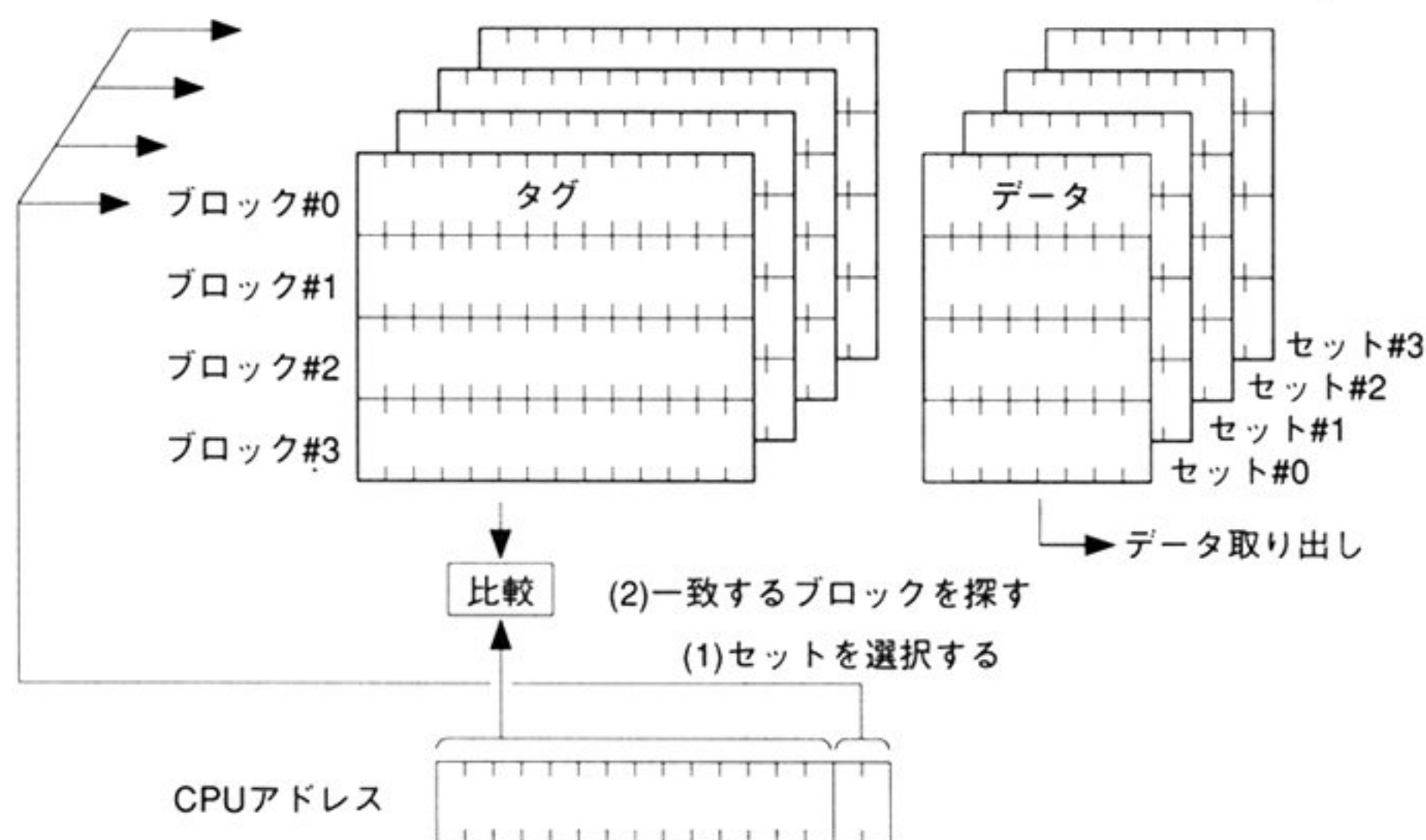
フルアソシエイティブ方式は，アドレス情報で直接ブロックを選択するのをやめ，キャッシュメモリのすべてのブロックのタグ情報とアドレス情報を比較するものです。一致するものがなければ，適当なものをキャッシュから追い出して新しいタグやデータを格納します。

どのブロックを追い出すかということに関してはいくつかの方法が考えられます。最も効果的なのは，将来使われることが最も少ないものから追い出すことです。しかし，将来を予測するのは難しいので，次善策として，アクセスされた時刻が最も古いものから追い出すという方法を使うのが一般的です。この方法を「LRU (Last Recently Used)方式」と呼びます。

フルアソシエイティブ方式のキャッシュの概要を図15に示します。この例では8個のブロックがあり，CPUは16ビットのアドレスを出力するものとしています。フルアソシエイティブ方式なので各ブロックのタグとCPUアドレスが比較され，一致するものがあればそのブロックのデータが引き出されます。

5・3・3 セットアソシエイティブ方式

フルアソシエイティブ方式では，キャッシュメモリの容量に比例してアドレスとタグの比較回路が増えていくことになるため，大きなキャッシュメモリを持たせるのはたいへんです。

●図16……セットアソシエイティブ方式(4ウェイセットアソシエイティブ)

これに対処するため，キャッシュメモリをいくつかのグループ（「セット」と呼びます）に分割し，アドレスの下位ビットでセットを選択し，選択されたセットについてはフルアソシエイティブ方式を行うようにする方法が考えられました。たとえば，キャッシュメモリが4Kバイトあっても，これを4つのセットに分けておけば，比較回路はキャッシュメモリが1Kバイトのときと同じ量ですむというわけです。

セットアソシエイティブ方式でセットの数がN個の場合,「Nウェイセットアソシエイティブ方式」という言い方をします。

図16にセットアソシエイティブ方式の例を示します。この例では16ブロックのキャッシュを4セットに分割しています。CPUのアドレスの下位2ビットで，この4セットのうちどのセットを使うかを選択し，アドレスの上位14ビットとタグが比較され，一致するものがあればデータが引き出されます。

キャッシュメモリの容量が数Kバイト以上の場合はほとんどセットアソシエイティブ方式が使われています。M68000ファミリーでも，MC68040の内蔵キャッシュメモリは4ウェイセットアソシエイティブ方式となっています。

5・3・4 セクタ方式

セクタ方式は，キャッシュのブロックをいくつかずつまとめた，「セクタ」と呼ばれる単位を作り，これにタグをつけておく方法です。セクタの選択はフルアソシエイティブ方式で行います。選択されたセクタのなかからブロックを選択する方法としてはダイレクトマップ方式を使

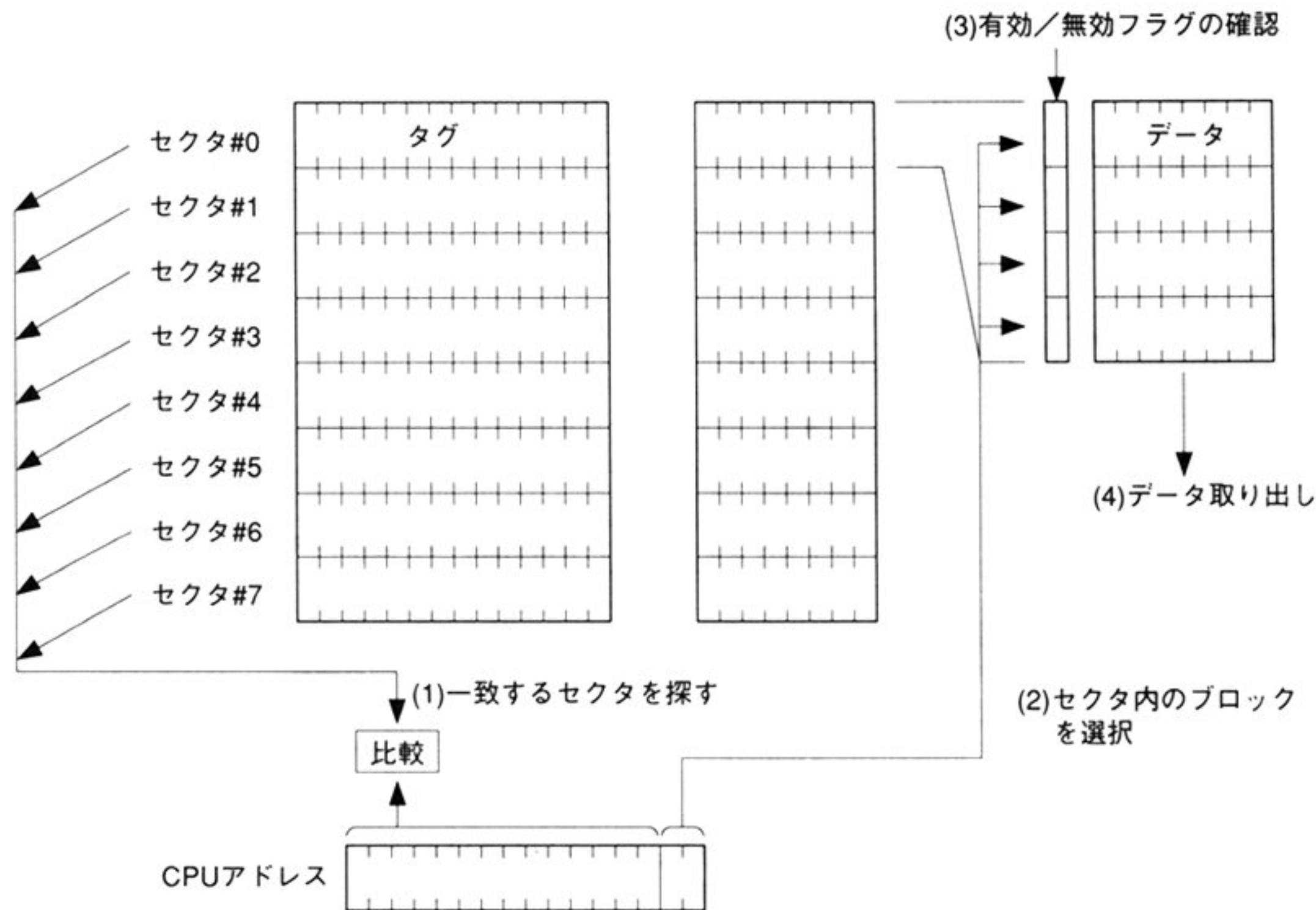
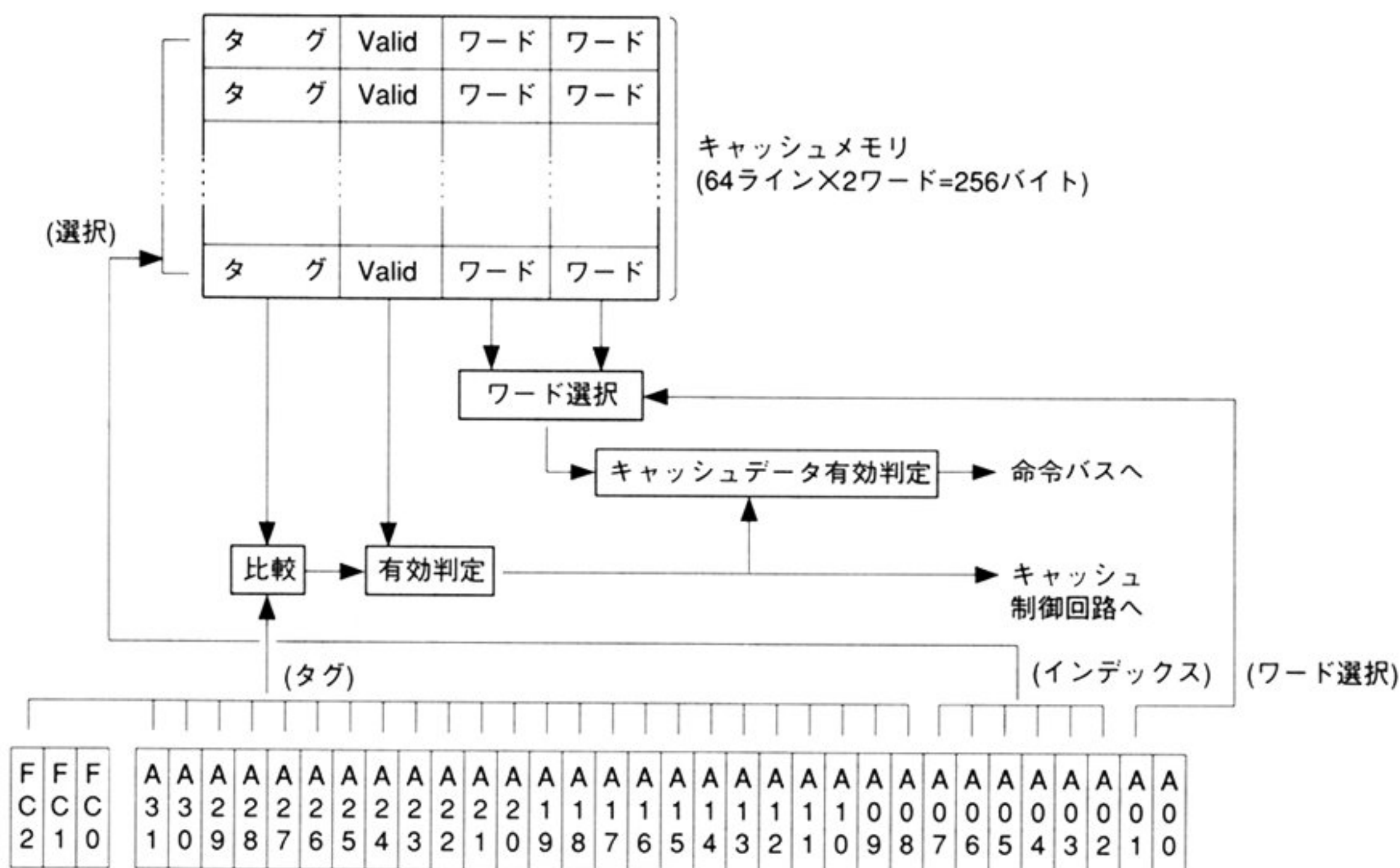
●図17……セクタ方式

用します。このため，各ブロックにはそのブロックが有効であるか否かを示すフラグが必要となります。

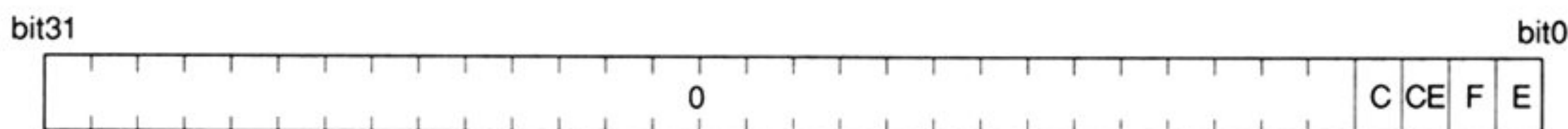
図17にセクタ方式の例を示します。この例ではキャッシュメモリは8つのセクタからなっており，各セクタは4ブロックのキャッシュからなっています。まず，アドレスの上位14ビットが各セクタごとにつけられたタグと比較され，一致するセクタがあれば，さらにそのセクタのなかから該当するブロックをアドレスの下位2ビットで選択します。このブロックに有効フラグが立っていればキャッシュヒットということになり，データが引き出されます。

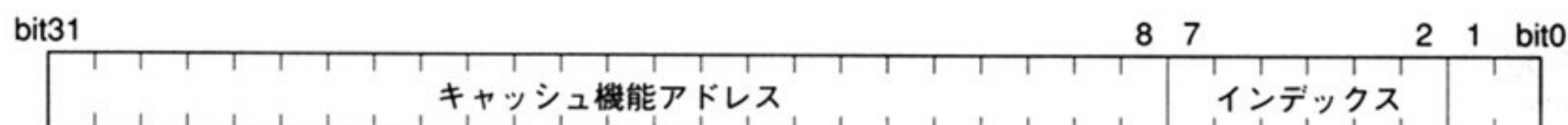
5・4 キャッシュメモリと関連レジスタ

MC68020/68030/68040の各CPUのキャッシュメモリの構成と制御レジスタのビット配置を図18～図23に示します。キャッシュメモリはMC68020/68030がダイレクトマップ方式，MC68040が4ウェイセットアソシエイティブ方式となっています。

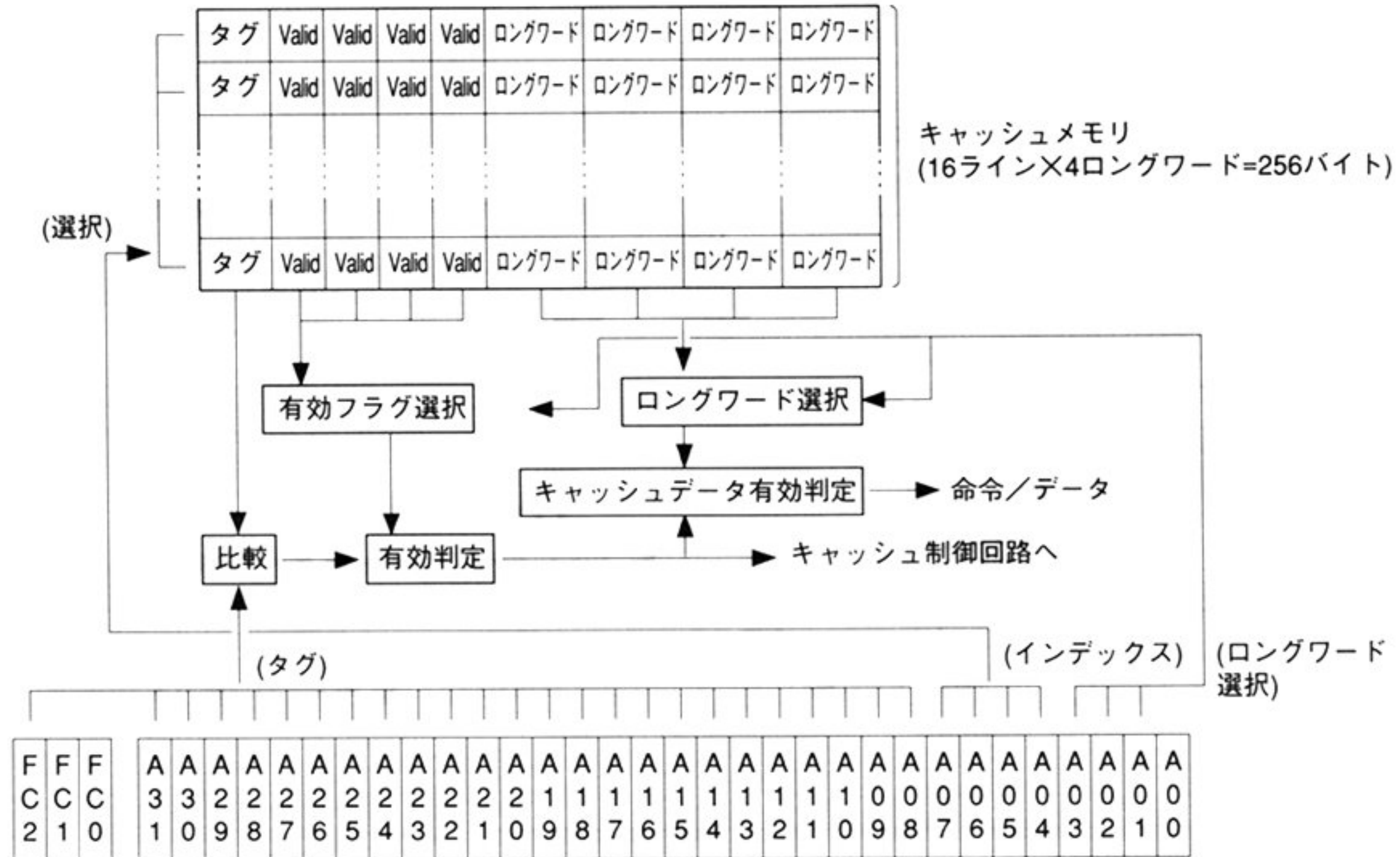
MC68020では，キャッシュメモリが働くのは命令コードだけでしたので，CACRのキャッシュ制御ビットは下位4ビットにイネーブル，フリーズ(凍結)，エントリクリア，クリアの4ビットを割り振っているだけでした。これに対しMC68030では，命令だけでなくデータ用にも

●図18……MC68020のキャッシュメモリの構成

●図19……MC68020のキャッシュ制御関係のレジスタ

MC68020のキャッシュコントロールレジスタ

C：キャッシュクリア
CE：キャッシュエントリクリア
F：キャッシュフリーズ
E：キャッシュイネーブル

MC68020のキャッシュアドレスレジスタ

256バイトのキャッシュメモリが実装されたため，ビット8～11に命令コード用と同様の制御ビットがデータキャッシュ用として追加されたほか，データ書き込み時にキャッシュミスした場合，キャッシュ内容を変更するか否かを制御するWAビットが追加されています。

また，MC68030ではキャッシュメモリに効率よくデータを取り込むためのバースト転送モードが追加されており，この動作を行うか否かの制御ビット（CBE/DBE）が設けられていま

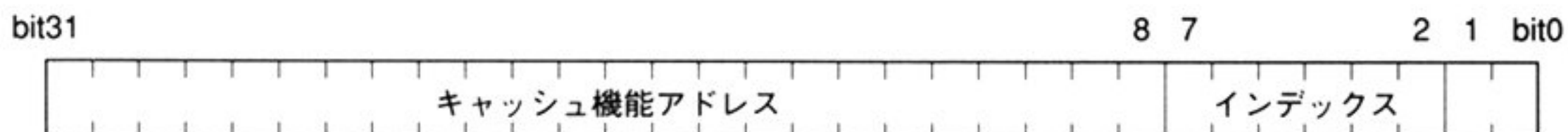
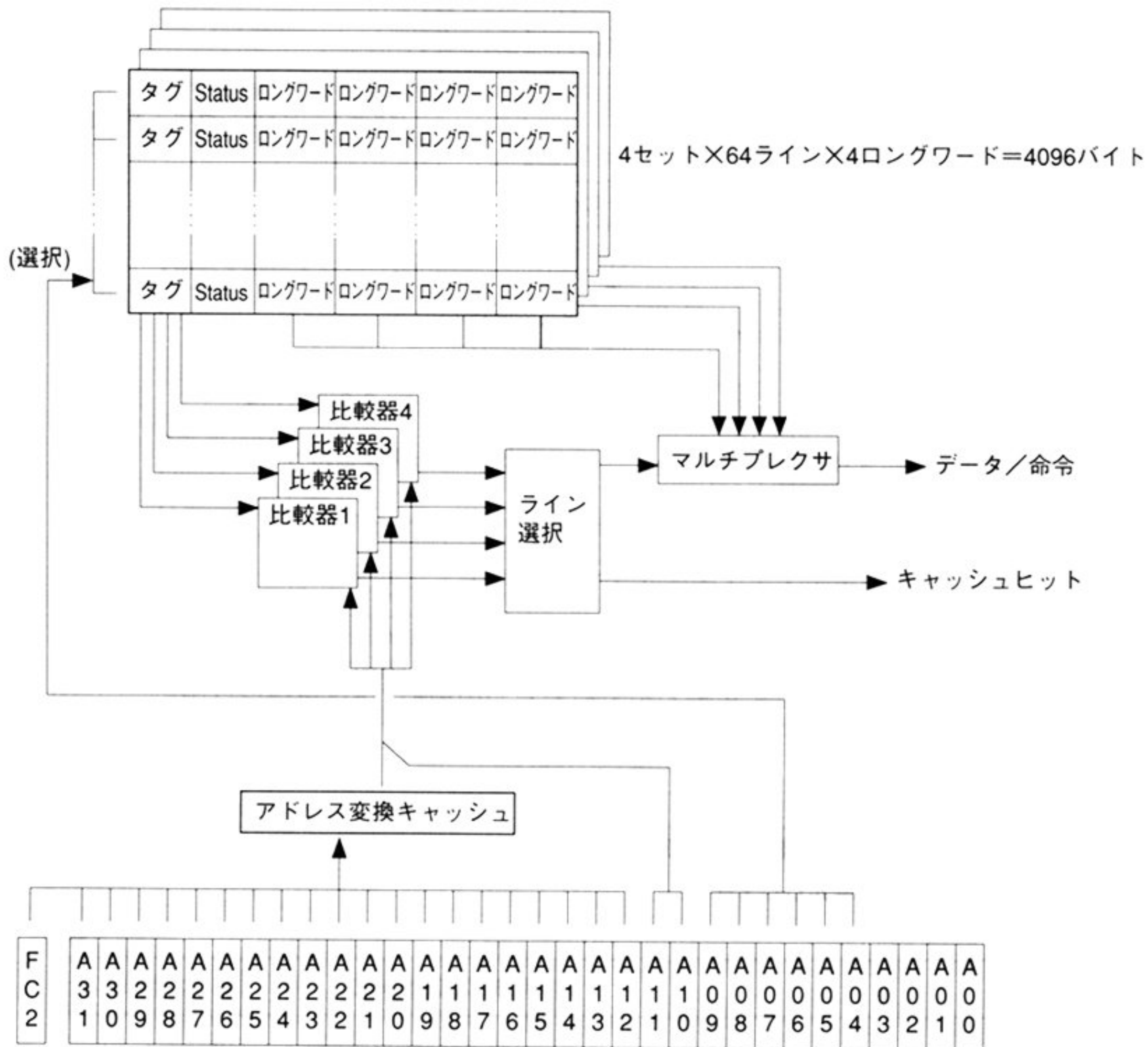
●図20……MC68030のキャッシュメモリの構成(命令／データとも同様)

●図21……MC68030のキャッシュ制御関係のレジスタ

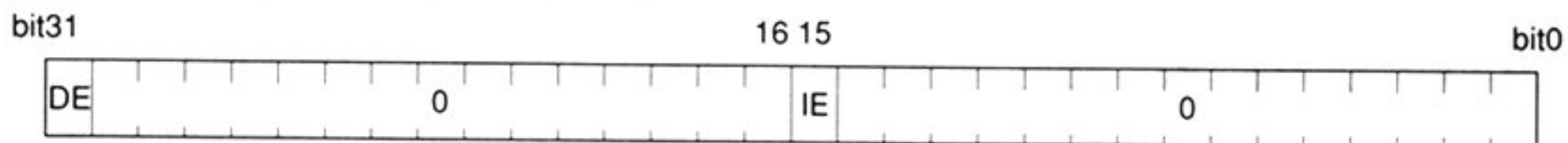
MC68030のキャッシュコントロールレジスタ

WA：ライトアロケート
DBE：データバーストイネーブル
CD：データキャッシュクリア
CED：データキャッシュエントリクリア
FD：データキャッシュフリーズ
ED：データキャッシュイネーブル
IBE：命令バーストイネーブル
CI：命令キャッシュクリア
CEI：命令キャッシュエントリクリア
FI：命令キャッシュフリーズ
EI：命令キャッシュイネーブル

MC68030のキャッシュアドレスレジスタ

bit31　8　7　2　1　bit0

キャッシュ機能アドレス	インデックス	

●図22……MC68040のキャッシュメモリの構成(命令／データとも同様)

●図23……MC68040のキャッシュ制御関係のレジスタ

MC68040のキャッシュコントロールレジスタ

DE：データキャッシュイネーブル
IE：命令キャッシュイネーブル

す。X68030では，外部回路がバースト転送モードに対応していないため，CBE/DBEビットの設定は意味を持ちません。

MC68040では，キャッシュメモリのメインテナンス用の命令が追加されたのにともない，

CACRはデータキャッシュ／命令キャッシュそれぞれのON/OFF制御ビットのみとなっています。キャッシュON/OFFの制御ビットの位置がMC68020/68030では未使用であったところに割り振られていることに注意してください。

5・5 書き込み時の動作方式

キャッシュメモリによりメモリリード動作は飛躍的に向上しましたが，メモリへの書き込み動作については最終的にはメモリにデータを書き込まざるを得ないため，リード動作ほど効果のある方法はありません。

現在使われている方法には，書き込み動作のときは必ずメモリにも書き込む「ライトスルー（ストアスルーやストアイミディエイトとも呼ばれることもあります）方式」と，書き込みを行ったアドレスがキャッシュメモリ内にあれば，いったんキャッシュメモリに書き込むだけですませておき，該当するブロックがキャッシュメモリから追い出されるときなどにはじめてメモリへの書き込みを行うようにする「ライトバック（ストアバックやスワップと呼ぶこともあります）方式」があります。ライトバック方式のほうがライトスルー方式よりも性能は高くなりますが，制御回路がかなり複雑になるのが欠点であるといえます。

5・6 キャッシュの一致化（スヌーピング）

キャッシュメモリはメインメモリのコピーを保持しており，キャッシュがヒットしている間はメインメモリに対するアクセスは行われません。このため，なんらかの理由でメインメモリの内容だけが書き換えられた場合には，キャッシュメモリとメインメモリの内容が一致しないことになります。この場合，キャッシュメモリの該当ブロックの内容を更新するか無効化して，強制的にメモリアクセスが行われるようにしておかなければなりません。この動作を「キャッシュのスヌーピング」と呼びます。

X68000/X68030シリーズでこの問題が起きるのは，キャッシュがヒットする領域に対してDMA転送が行われた場合です。キャッシュがヒットしている領域にディスクなどからデータを読み込んだ場合，DMA転送が完了してもキャッシュがヒットし続けるため，メモリ上のデータが読み出せないことになります。

MC68020/MC68030はキャッシュメモリを内蔵しましたが，いずれもCPUにはスヌーピング機構が実装されなかったため，このような問題が発生してしまいます。このため，X68030はIOCSのなかでキャッシュを無効化し，一致化を図るような方法を使っています。

他社のCPUを見てみると，このような方法をとらざるを得ないものは少数派で，通常はキャッシュメモリのスヌーピングはハードウェアで自動的に行われるようになっています。キャッシュメモリの制御回路がCPU以外のものからのメモリアクセスを監視し，スヌーピングを行うわけです。

MC68040では，CPUにスヌーピングを行うための信号線が追加され，ハードウェアでスヌーピングを行うことができるようになっています。

6 MMU

MMU（メモリマネージメントユニット）は，CPUがメモリアクセスを行うとき，CPUが出力したアドレスをあらかじめ与えられたテーブルの値に従って変換したり，メモリアクセスに制約を加えるものです。MMUを使うと，ハードディスクなどをあたかもメモリの一部であるかのように見せかけることができるようになり，実際に存在する容量以上のメモリを必要とするようなアプリケーションでも動作させることが可能になります。このような手法で得られたメモリを「仮想記憶」と呼びます。

M68000ファミリーのCPUでは，MC68010から外付けのMMUへの対応ができるようになり，MC68020と同時に外付けのMMUコプロセッサであるMC68851が発表されました。MC68851は，MC68881（数値演算プロセッサ）と同様，MC68020のコプロセッサとして動作するほか，I/OデバイスとしてMC68020以外のCPUにも接続できるようになっています。MC68030ではMC68851の機能を縮小したものが内蔵され，MC68040ではさらに機能を絞り込んだMMUが内蔵されています。

X68000のCPUはMMUがサポートできないMC68000です。また，X68030に使用されたCPU，MC68EC030はMC68030からMMUを外したものです。このため，X68000/X68030シリーズでは仮想記憶を扱うことができません。MC68EC030はMC68030とピンコンパチブルですので，そのままMC68030と差し替えることもできますが，OSであるHuman68k側が仮想記憶をサポートしていませんので，差し替えてもなんら変わることはありません。

CRTC

CRTCには，設定値と動作タイミングの関係など非公開となっている部分があります。ここでは，筆者が独自に調べた内容をもとに『Inside X68000』で触れられなかった，動作タイミングの詳細などについて説明します。

1 CRTCの画面モード設定のからくり

画面の縦横のドット数や水平周波数など，基本的なパラメータはCRTCのレジスタR00～R07とR20，およびシステムポートのレジスタ#4のHRLビットによって決められます。マニュアルなどで公開されている画面モードと，それぞれの画面モードでのレジスタの設定値については，計算方法とともに『Inside X68000』に示しましたが，HRLビットとの連携を含めた細かな内容については触れられなかったので，ここで補足しておきます。

なお，以下の内容は筆者が個人的に調べたもので，実際にX68000やX68030がこのとおりに作られていることを保証するものではありません。将来の機種追加などによって変更される可能性もありますので，使用にあたっては注意してください。

2 基本タイミングの制御

2・1 レジスタへの設定値の意味

まず，R00～R07の計算方法として示されている式を図1に示します。R00～R03が水平方向のタイミング，R04～R07が垂直方向のタイミング制御になっています。これらの式を見ていると，同じようなパラメータが何度も登場していることに気がつきます。水平方向では分子の(水平表示ドット数)と分母の(データ表示期間)×8が，垂直方向では分母の(水平同期期間)がそれにあたります。

垂直方向については，要するに目的とするタイミングが水平同期期間の何倍にあたるかを計算しているだけなので簡単ですが，水平方向のタイミングは少々複雑に見えます。そこで，まず共通になっているパラメータに注目してR00の計算式を変形してみます。

(R00＋1)×8×(水平データ表示期間)÷(水平表示ドット数)＝(水平同期期間)

●図1……CRTC　R00～R07の設定値の算出法

$$\mathrm{R00} = \frac{(\text{水平同期期間})\times(\text{水平表示ドット数})}{(\text{データ表示期間})\times 8} - 1$$

$$\mathrm{R01} = \frac{(\text{水平同期パルス幅})\times(\text{水平表示ドット数})}{(\text{データ表示期間})\times 8} - 1$$

$$\mathrm{R02} = \frac{((\text{水平同期パルス幅})+(\text{水平バックポーチ}))\times(\text{水平表示ドット数})}{((\text{データ表示期間})\times 8} - 5$$

$$\mathrm{R03} = \frac{((\text{水平同期期間})-(\text{水平フロントポーチ}))\times(\text{水平表示ドット数})}{(\text{データ表示期間})\times 8} - 5$$

$$\mathrm{R04} = \frac{(\text{垂直同期期間})}{(\text{水平同期期間})} - 1$$

$$\mathrm{R05} = \frac{(\text{垂直同期パルス幅})}{(\text{水平同期期間})} - 1$$

$$\mathrm{R06} = \frac{(\text{垂直同期パルス幅})+(\text{垂直バックポーチ})}{(\text{水平同期期間})} - 1$$

$$\mathrm{R07} = \frac{(\text{垂直同期期間})-(\text{垂直フロントポーチ})}{(\text{水平同期期間})} - 1$$

左辺の(水平データ表示期間)÷(水平表示ドット数)というのは何を示しているのでしょうか。水平方向のデータ表示期間を，表示されているドット数で割るのですから，水平方向の1ドットあたりの表示時間を示していることにほかなりません。1ドットあたりの表示時間を決定しているクロックを「ドットクロック」と呼んでいます。1ドットあたりの表示時間というのは，ドットクロックの周期にあたります。つまり，先ほどの式はドットクロックの周期を使うと，

(R00＋1)×8×(ドットクロック周期)＝(水平同期期間)

となります。

つまり，R00への設定値というのは，水平同期期間がドットクロック周期の8倍のさらに何倍にあたるかを指定するものであることがわかります。ここで出てきた8という定数はほかのレジスタでも同じなので，X68000のCRTCは8×(ドットクロック周期)，すなわち8ドット分の表示時間をもとに動作しているのではないかという予測ができます。さらに8×(ドットクロック周期)を，単に(8ドットクロック周期)と置き換えてしまうことにします。すると，

(R00＋1)×(8ドットクロック周期)＝(水平同期期間)

と，ずいぶんすっきりした式が得られます。同じようにR01～R03についても，

(R01＋1)×(8ドットクロック周期)＝(水平同期パルス幅)
(R02＋5)×(8ドットクロック周期)＝(水平同期パルス幅)＋(水平バックポーチ)
(R03＋5)×(8ドットクロック周期)＝(水平同期期間)－(水平フロントポーチ)

と変形できます。

つまり，8ドットクロックを基本にして，それぞれのタイミングが何クロック周期分になっているかを設定していたというわけです。

2・2 ドットクロックの選択のしくみ

この基本となっている8ドットクロックの選択に関与していると思われるのは，R20とシステムポート#4のHRLビットです。R20，およびシステムポート#4のビット配置を図2と図3に示します。R20のうち8ドットクロックに関与していると考えられるビットは，HF(ビット

●図2……CRTC　R20のビット配置

アドレス:$E80028

●図3……システムポート#4のビット配置

アドレス:$E8E007

4)，およびHD（ビット0,1）です。

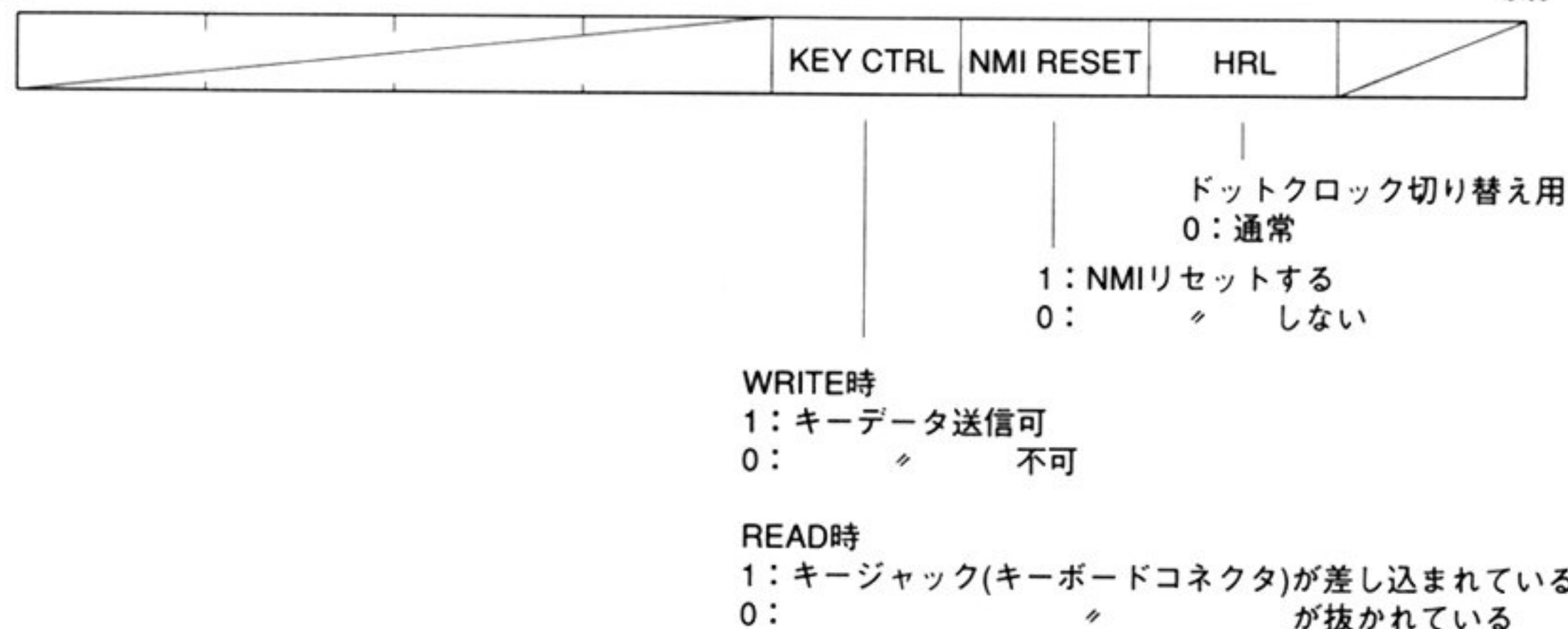
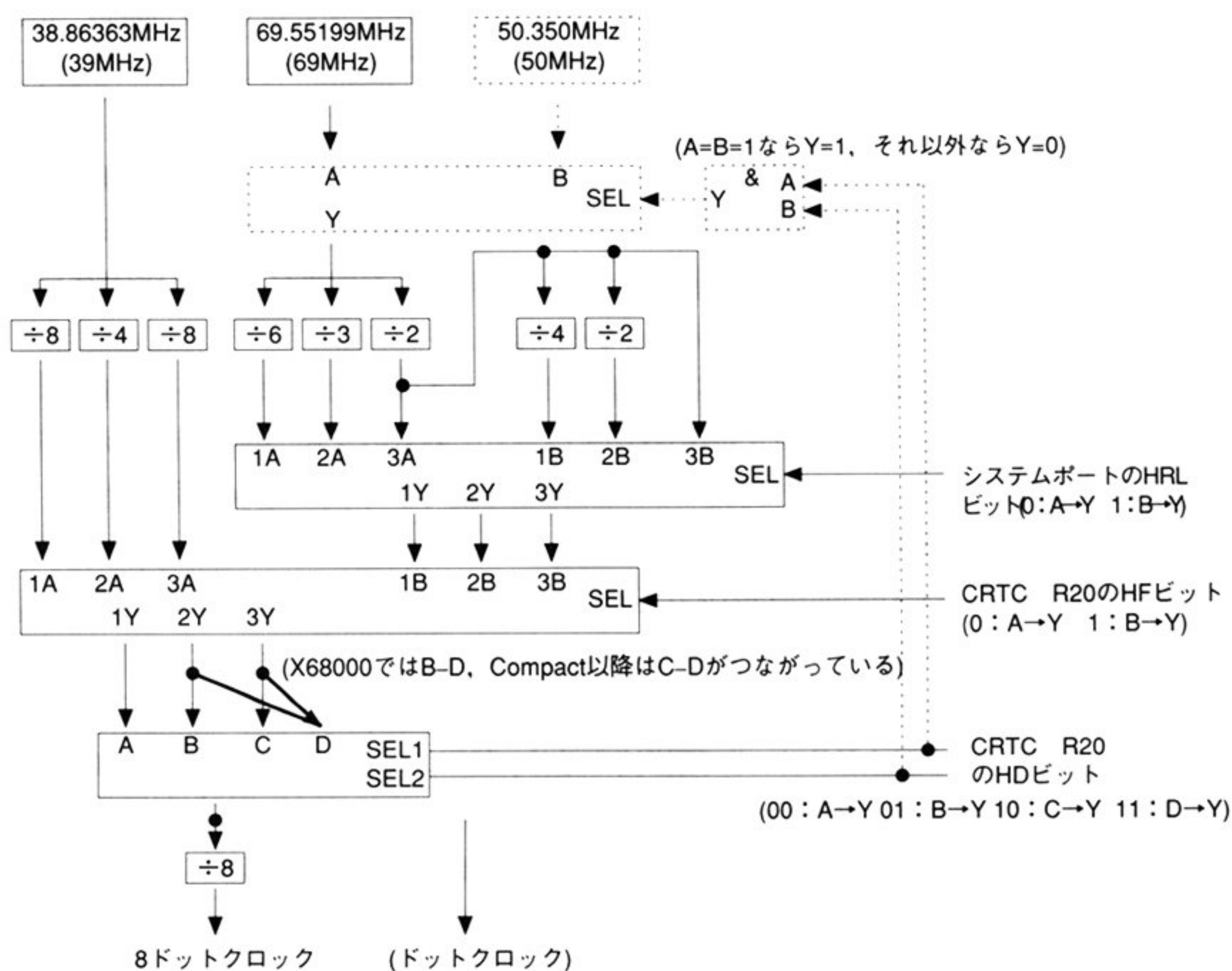



図4にこれらの値と，実際にCRTに出力されている波形，およびX68000/X68030の回路図などから推定したクロック切り替えの機構を示します。X68000の回路図を見ると，ドットクロック源となっていそうなクロックとして39MHzと69MHzの発振器が，X68000 Compact XVIとX68030には39MHzと69MHzに加え，50MHzのクロックがあります。実際にCZ-600C（元祖タイプ）とCZ-500C（X68030）を分解してみると，CZ-600Cでは69.55199MHzと38.86363MHz, CZ-500Cでは69.551MHzと38.863MHz, 50.350MHzの発振器が取り付けられていました。ここでは簡略化して69MHz, 39MHz, 50MHzと呼ぶことにしておきます。

●図4……ドットクロック切り替えの仕組み

＊……部分はX68000 Compact XVI, X68030のみ。

これらのうち, 39MHzが標準解像度モード用, 69MHzは高解像度モード用として使われています。50MHzはX68000 Compact XVIとX68030のIOCSにある非公開画面モード19(640×480ドット)で使用されています。50.350MHzという周波数は, PC/AT互換機でよく使われているグラフィックアダプタであるVGA(640×480ドット)のドットクロック25.175MHzのちょうど2倍にあたります。画面モード19は, VGAとの互換性を重視したのだと思われます。

X68000 Compact XVIで追加された50MHzの発振器は69MHzの発振器と切り替えて使うようになっています。この切り替えは, R20のHDビット(ビット0,1)で行われ, '11'のときに50MHzが, それ以外のときは69MHzが選択されるようになっています(図4の点線部分)。X68000 Compact XVI以前の機種にはこのようなものはありませんので, 69MHzがそのまま使われることになります。

図のなかで1Aや1B, 1Yなどと書かれているのは切り替えスイッチのような働きをするもの

で,「セレクタ」と呼ばれます。SELが0のときには'1Y', '2Y', '3Y'がそれぞれ'1A', '2A', '3A'とつながり, SELが'1'のときには'1B', '2B', '3B'と接続されたことになります。同様にSELが2本あるときには, '00'なら'A', '01'で'B', '10'で'C', '11'で'D'と'Y'が接続されたのと同じような動作になります。

次に, これらの周波数から各画面モード用のクロックが生成されます。まず, 高解像度モード用のクロックとして69MHz/50MHzのクロックを2分周（周波数を2分の1にすること), 3分周, 6分周したものが作られます。図中"÷8"などと書かれた四角が, この分周動作をするものです。"÷8"とあれば, 8分周することを示します。

69MHz/50MHzを2分周したものから, さらに2分周, 4分周した（つまり, 69MHz/50MHzを4分周, 8分周した）クロックが作られています。これらのうち, いずれを使うかがシステムポートのHRLビットで選択されます。HRLビットが0なら2, 3, 6分周の組が, 1なら2, 4, 8分周の組が使われます。

標準解像度モードのほうの39MHzのクロックは512ドットモード, 256ドットモード用としてそれぞれ4分周, 8分周したクロックが生成されています。

次に標準解像度モード用のクロックと高解像度モード用のクロックから生成されたクロック群のいずれを使うかがR20のHFビットで選択されます。HFビットが'1'だと69MHz/50MHzから生成されたクロックが, '0'だと39MHzから生成されたクロックが使用されます。

そして, 最後に選択されてきたクロック群のうち, どれを使うかがR20のHDビットで選択されます。この部分動作は, X68000とX68000 Compact XVI/X68030とでは少し違っています。HDビットが'11'になっていると, X68000は'01'と同じことになりますが, X68000 Compact XVIやX68030では'10'と同じになります。こうやって選択されたクロックがドットクロック（画面上の1ドットあたりのクロック）になります。

CRTCのタイミング計算などに使われる8ドットクロックは, このクロックをさらに8分周したものとなります。

2・3 ドットクロックの値

少々複雑ではありましたが, 発振器の周波数とクロック切り替えの機構や設定値と分周比の関係がわかったので, ここから実際の8ドットクロックの値を算出することができます。図5にレジスタへの設定値と8ドットクロックの値の関係をまとめてみました。数値は, 実際に基板についていた発振器の周波数から計算した値（69.55199MHz/38.86363MHz/50.350MHz)です。( )内の数値が, 発振器の周波数が回路図どおりの値(69MHz/39MHz/50MHz)であるとして計算したときの値になっています。

●図5……レジスタの設定値と8ドットクロック周期の関係

システムポート#4	CRTCレジスタ R20					備　考
HRL	HF	00	01	10	11	
0	0	1.64678 (1.64103)	0.82339 (0.82051)	1.64678 (1.64103)	1.64678 (1.64103)	15KHzモード用　256/512 ドット
0	1	0.69013 (0.69565)	0.34507 (0.34783)	0.23004 (0.23188)	0.31778 (0.32000)	31KHzモード用　256/512/768 ドット
1	0	0.92017 (0.92754)	0.46009 (0.46377)	0.23004 (0.23188)	0.31778 (0.32000)	24KHzモード用　192/384/768 ドット

単位: μS

数値は，どれも小数点以下５桁目で四捨五入しています。もっと高い精度が必要ならクロック周波数から割算をして求めればよいでしょう。

2・4 CRT側のタイミング

さて一方，CRT側に与える標準的なタイミング，すなわち，先ほど変形した式の右辺の値はどうなっているのでしょう。15KHzモードと31KHzモードのときの値については公開されて

●図6……各水平周波数時のディスプレイ信号タイミング

水平周波数	種別	表示期間	同期期間	同期パルス幅	バックポーチ	フロントポーチ	単位
		信号タイミング					
31KHz	水平	22.0842 (22.2609)	31.7460 (32.0000)	3.4507 (3.4783)	4.1408 (4.1739)	2.0704 (2.0870)	μS
	垂直	16.2540 (16.3840)	18.0317 (18.1760)	0.1905 (0.1920)	1.1111 (1.1200)	0.4762 (0.4800)	mS
31KHz (V)	水平	25.4220 (25.6000)	31.7776 (32.0000)	3.8133 (3.8400)	1.9067 (1.9200)	0.6356 (0.6400)	μS
	垂直	15.2532 (15.3600)	17.2552 (17.3760)	0.0636 (0.0640)	1.0169 (1.0240)	0.9215 (0.9280)	mS
24KHz	水平	29.4456 (29.6812)	40.4877 (40.8116)	3.6807 (3.7101)	4.6009 (4.6377)	2.7605 (2.7826)	μS
	垂直	17.1668 (17.3401)	18.8268 (18.9773)	0.3239 (0.3265)	1.0122 (1.0203)	0.3239 (0.3265)	mS
15KHz	水平	52.6971 (52.5128)	62.5778 (62.3590)	3.2936 (3.2821)	4.9404 (4.9231)	1.6468 (1.6410)	μS
	垂直	15.0187 (14.9662)	16.2702 (16.2133)	0.1877 (0.1871)	0.8761 (0.8730)	0.1877 (0.1871)	mS
VGA (参考)	水平	25.4220	31.7776	3.8133	1.9464	0.5958	μS
	垂直	15.2532	16.6832	0.0636	1.0169	0.3496	mS

＊数値は発振器の周波数を69.55199MHz, 38.86363MHz, 50.350MHzとして算出した値。
＊( )内の数値は発振器の周波数が69MHz, 39MHz, 50MHz（回路図中の公称値）とした場合の計算値。

いたので『Inside X68000』で表にしてまとめましたが，本当にあの値どおりになるのかどうかをレジスタへの設定値から逆算してみました。計算結果を図6に示します。数値は，実際の基板についていた発振器の周波数をもとに計算したものです。念のため，( )内に回路図どおりの値を使った場合の計算値も併記しておきましたので参考にしてください。

24KHzモードについてはCRT側のタイミングは公開されていませんが，画面モード設定のIOCSコールで非公開となっている画面モード18 (1024×848ドットモード) の設定値を使いました。

また，31KHz(V)となっているのは640×480ドットモードのときの値です。数値は，X68030のIOCSコールの非公開画面モード19の設定値から求めました。X68000で通常使われている31KHzモードの値と比べると，水平のバックポーチとフロントポーチがかなり少なく，また垂直のフロントポーチが大きくなっています。このため，X68000用のディスプレイで表示すると，横方向は画面の外まではみ出し，縦方向は下が切られたようになるため，やや横長の表示になります。念のため，PC/AT互換機でよく使われているグラフィックアダプタであるVGAのタイミングを実測した結果も併記しておきます。なかなかよく似たタイミングになっています。

2・5 水平方向のタイミング操作

8ドットクロックとディスプレイ側のタイミングがわかれば，CRTCのレジスタの設定次第でいろいろな画面モードが作れることがわかります。たとえば，15KHzモード時の512ドット用とされている8ドットクロックを選択しておき，31KHzモードのタイミングにあうようにレジスタの値を調整してみます。垂直方向のドット数をとりあえず512とすると，各レジスタの設定値は次のようになります。

```
R0  =   39($0027)
R1  =    3($0003)
R2  =    4($0004)
R3  =   31($001F)
R4  =  567($0237)
R5  =    5($0005)
R6  =   40($0028)
R7  =  552($0228)
R20 =  769($0301)
```

このときの横方向のドット数は，(R3－R2)×8で算出できます。R3－R2は，水平同期期間から同期パルス幅，フロントポーチ，バックポーチの3つを引いた分，すなわち，水平表示期

間が8ドットクロック周期の何倍であるかを示していることになります。したがって，R3－R2を計算して，それを8倍すれば水平方向の表示ドット数が算出できます。

この計算結果にあてはめると，表示ドット数は横216ドットとなります。縦方向は31KHzモードの標準値のままですので512ドット，つまり，216×512ドットモードの画面になります。

3 垂直方向の動作モード選択

水平方向は解読できたので，次に垂直方向の動作について調べてみます。垂直方向のタイミングは水平同期期間の時間をもとに作成されていますので，水平方向と同様にさまざまなタイミングを作ることも可能です。ただ，残念なことにX68000の専用ディスプレイは垂直同期期間が約16mS（約60Hz）の信号にしか対応できませんので，標準値から大幅に変えてしまうと画面が流れて何を表示しているのかがわからなくなります。もしマルチシンクモニタを使用されているのであれば，垂直タイミングを変えた画面モードを作ってみるとよいでしょう。

その不自由さの代償というわけでもないのでしょうが，X68000/X68030では垂直方向の動作モードとして，ノン・インターレース，インターレース，二度読みの3つのモードをサポートしており，これらを切り替えることで表示ドット数を変えられるようになっています。

ノン・インターレースというのは，31KHzモードでの縦512ドットモードや15KHzモードでの256ドットモードに相当する，最も一般的な動作モードに相当します。

インターレースは，奇数回目と偶数回目の画面描画の際に異なるデータを送ることで垂直方向の表示ドット数を2倍にする方法です。この場合，あるドットが書き直される周期はノン・インターレースのときの半分になりますので，画面がちらついて見えます。

二度読みは，奇数本目と偶数本目のラインに同じデータを表示する動作モードです。垂直方向のドット数がノン・インターレースのときの半分になります。

●図7……HL, VDビットの設定と垂直方向の動作モード

HL	VD	動作モード
0	00	ノン・インターレース
	01	インターレース
	10	非公開（インターレース）
	11	非公開（インターレース）
1	00	二度読み
	01	ノン・インターレース
	10	非公開（インターレース）
	11	非公開（インターレース）

これらの動作はR20のHLビット，およびVDビットで決められます。この関係を図７に示します。HLが'0'のときは二度読み動作はできないようになっています。HLが'1'のときにはVDビットに'10'を設定することでインターレースモードで動作します。この設定は非公開扱いですので将来変更される可能性がないとはいいきれませんが，IOCSコールの画面モード18でも使っているところを見ると，公然の秘密といってもよいのかもしれません。

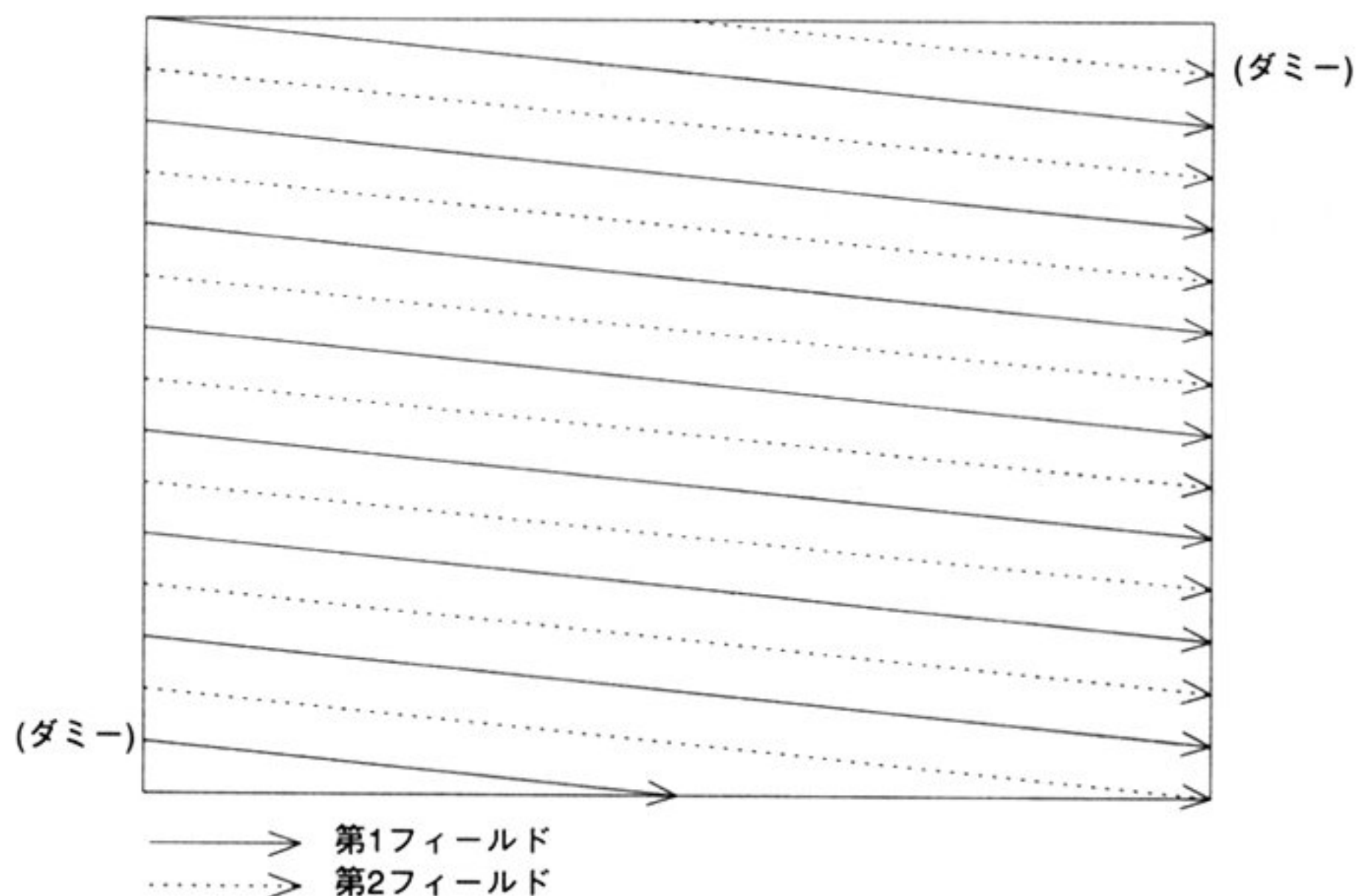
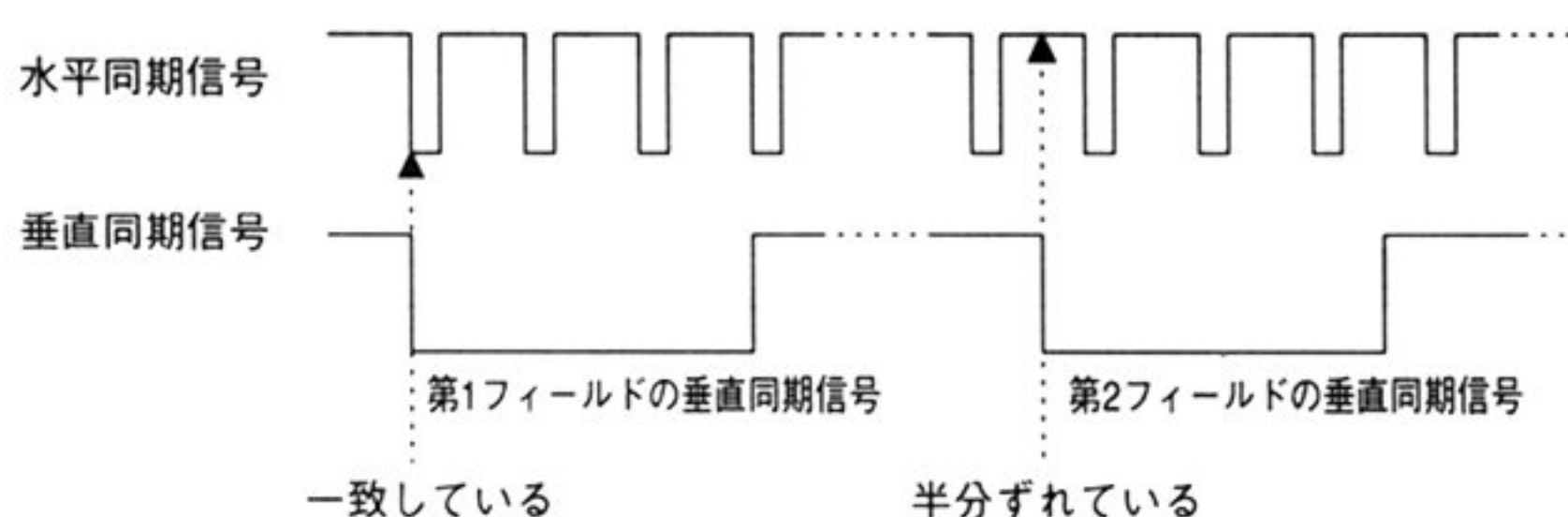
COLUMN

インターレースモードと水平・垂直同期信号

水平同期信号は，レジスタへの設定値どおり定周期のパルス信号が出力されますが，垂直同期信号は，インターレースモードのときとノン・インターレースモードのときではタイミングが少々異なります。

ノン・インターレースモードのときは，単純にディスプレイの一番上のラスターから一番下のラスターまでを表示して上に戻るという動作を繰り返しますので，ディスプレイに与える同期信号も単純です。垂直同期信号の立ち下がり／立ち上がりは必ず水平同期信号の立ち下がりに同期しており，垂直同期信号の幅もレジスタの計算式どおりになります。

一方，インターレースモードでは偶数番目（０，２，４……）のラスターで作られる画面（第１フィールド）と，奇数番目（１，３，５……）のラスターで作られる画面（第２フィールド）を交互に表示します（図８を参照してください)。このため，ディスプレイ側でどちらを描画するのかを判別できるようにする必要があります。インターレースモードでは，垂直同期期間を水平同期期間の半分だけ引き延ばすことで，垂直同期と水平同期の

●図8……インターレースモード時の描画動作

●図9……インターレース動作時の垂直同期信号

位置関係をフィールドごとにずれるように対処しています。第1フィールドの開始時の垂直同期信号は，水平同期信号の立ち下がりに同期しており，第2フィールドでは水平同期信号の立ち下がりから水平同期期間の半分だけずれた位置に垂直同期信号がくることになります(図9)。つまり，実際のディスプレイ側の動作は，第1フィールドは画面の左上から開始され，右下までいった後，さらに画面の下の左半分まで走査したところでようやく垂直表示期間が終了し，同期信号によって第2フィールドに移行することになります。

第2フィールドは，水平同期信号がきて半分たってから垂直表示期間になりますので，画面上では一番上のラインは右半分だけ走査され，次のラスターから画面表示が始まります。

第1フィールドの最後と第2フィールドの最初に付加されるこれらの期間は，画面データの表示を行いませんので，単なるダミーの期間になります。

インターレースモードではこのような動作をするため，実際の垂直同期期間はレジスタの設定値に0.5を加えた値になっています。

4 特殊画面モード計算プログラム

X68000には公開されていないさまざまな画面モードを作成できる可能性があることがわかりましたが，いちいち電卓を叩いて計算するのは面倒です。そこで，8ドットクロックや垂直方向の動作モードなどを選択するだけでR0～R7，およびR20への設定値を計算するプログラムを作ってみました。

このプログラムではディスプレイ側のタイミングを5種類のなかから選択するようにしていますが，ディスプレイ側は多少この値からはずれていても追従できますし，また，表示期間を減らして，その分フロントポーチやバックポーチを増やすといった方法を使えば，実際の表示ドット数を加減することもできます。設定したいドット数のモードが見つからないときは近い

値のものを選んで調整すればよいでしょう。CRT側のタイミングデータとしては15KHz, 24 KHz, 31KHzの標準的な値のほかに，640×480ドットモード時，およびVGAの標準的な値も加えておきました。

●リスト　CRTCレジスタ設定値計算プログラム

```
/*
 *  ＣＲＴＣレジスタ設定値計算プログラム
 *
 */

#include <doslib.h>

void main();
void get_value(char *, unsigned int *, unsigned int, unsigned int);
void cal_regs(void);
void dsp_regs(void);
void set_regs(void);

float dot_clock8[3][4] = {           /* 上位:- [HRL:ｼｽﾃﾑﾎﾟ-ﾄ]*2+[HF:R20] */
                        /* 下位:- [HD:R20]           */
                        /*                  256 512 768    */
    1.64678, 0.82339, 1.64678,1.64678,  /* Low-Mode    8*1/39 *8  *4  *8    */
    0.69013, 0.34507, 0.23004,0.31778,  /* High-Mode   8*1/69 *6, *3, *2    */
    0.92017, 0.46009, 0.23004,0.31778   /* Middle-Mode 8*1/69 *8  *4  *2    */
};

float freq_data[5][10] = {
  22.0842,  31.7460, 3.4507, 4.1408,  2.0704,   /* HDSP,HSYNC,HP,HBP,HFP (31KHz)
*/
  16.2540,  18.0317, 0.1905, 1.1111,  0.4762,   /* VDSP,VSYNC,VP,VBP,VFP      */
  25.4220,  31.7776, 3.8133, 1.9067,  0.6356,   /* HDSP,HSYNC,HP,HBP,HFP (31KHz)
*/
  15.2532,  17.2552, 0.0636, 1.0169,  0.9215,   /* VDSP,VSYNC,VP,VBP,VFP（Ｖ）
*/
  25.4220,  31.7776, 3.8133, 1.9464,  0.5958,   /* HDSP,HSYNC,HP,HBP,HFP (31KHz)
*/
  15.2532,  16.6832, 0.0636, 1.0169,  0.3496,   /* VDSP,VSYNC,VP,VBP,VFP（ＶＧＡ）
*/
  29.4456,  40.4877, 3.6807, 4.6009,  2.7605,   /* HDSP,HSYNC,HP,HBP,HFP (24KHz)
*/
  17.1668,  18.8268, 0.3239, 1.0122,  0.3239,   /* VDSP,VSYNC,VP,VBP,VFP
*/
  52.6971,  62.5778, 3.2936, 4.9404,  1.6468,   /* HDSP,HSYNC,HP,HBP,HFP (15KHz)
*/
  15.0187,  16.2702, 0.1877, 0.8761,  0.1877    /* VDSP,VSYNC,VP,VBP,VFP
*/
};

unsigned int    hl,clk,freq,scanmode,colmode,size;
unsigned int    yesno;
```

```
unsigned int    ir[8],ir20;
unsigned int    hres,vres;
float       fr[8];
unsigned char   keybuf[128];
int     stack;


volatile unsigned short *crtc = (unsigned short *)0xe80000;
volatile unsigned short *vcon = (unsigned short *)0xe82400;
volatile unsigned short *scon = (unsigned short *)0xeb080a;
volatile unsigned char  *sysp = (unsigned char  *)0xe8e007;

void main()
{
    /* [HRL:ｼｽﾃﾑﾎﾟｰﾄ]*2+[HF:R20]の選択  */
    get_value("ﾄﾞｯﾄｸﾛｯｸ   (1:Low   2:High     3:Middle                 ) ",&
hl ,1,3);
    /* [HD:R20]の選択             */
    get_value("ﾄﾞｯﾄｸﾛｯｸ   (1:256   2:512      3:768        4:special       ) ",&
clk,1,4);
    if ((hl == 0) && (clk == 2)) {
        printf("Lowモードでは256とおなじです。\n");
        clk = 0;
    }
    get_value("水平周波数 (1:31KHz 2:31KHz(V) 3:31KHz(VGA) 4:24KHz 5:15KHz ) ",&
freq,1,5);
    do {
        get_value("走査モード (1:ﾉﾝ･ｲﾝﾀｰﾚｰｽ  2:ｲﾝﾀｰﾚｰｽ  3:二度読み              )
 ",&scanmode,1,3);
    } while ((hl == 0) && (scanmode == 2));
    get_value("実画面ｻｲｽﾞ  (1:512*512ﾄﾞｯﾄ 2:1024*1024ﾄﾞｯﾄ                 ) ",&
size,1,2);
    get_value("色数       (1:16色        2:256色    3:65535色             ) ",&
colmode,1,3);
    cal_regs();
    dsp_regs();
    get_value("\nCRTCに設定しますか?    (1:No   2:Yes                ) ",&
yesno,1,2);
    if (yesno)
        set_regs();
}

void get_value(s,i,l,h)
    char        *s;
    unsigned int    *i,l,h;
{
    do {
        printf("%s",s);
        if (!scanf("%d",i)) {
            printf("\n");
            scanf("%s",keybuf);
        }
    } while((*i < l)||(*i > h));
    (*i)--;
```

```
}

void cal_regs()
{
    unsigned int    i,hilo;
    float        dot_clk,hsync;
    dot_clk = dot_clock8[hl][clk];
    hsync   = freq_data[freq][1];
    fr[0] = (freq_data[freq][1]/dot_clk)-1.0;
    fr[1] = (freq_data[freq][2]/dot_clk)-1.0;
    fr[2] = ((freq_data[freq][2]+freq_data[freq][3])/dot_clk)-5.0;
    fr[3] = ((freq_data[freq][1]-freq_data[freq][4])/dot_clk)-5.0;
    fr[4] = (freq_data[freq][6]*1000.0/hsync)-1.0;
    fr[5] = (freq_data[freq][7]*1000.0/hsync)-1.0;
    fr[6] = ((freq_data[freq][7]+freq_data[freq][8])*1000.0/hsync)-1.0;
    fr[7] = ((freq_data[freq][6]-freq_data[freq][9])*1000.0/hsync)-1.0;
    for (i=0; i<8; i++) {
        if (fr[i] < 0.0) {
            printf("¥n***ERROR :- R%d (%f)¥n¥07",i,fr[i]);
            continue;
        }
        ir[i] = (unsigned int)(fr[i]+.5);
    }
    ir[0] |= 1;
    if (colmode == 2)
        colmode = 3;
    if (hl >= 1)
        hilo = 1;
    else    hilo = 0;
    ir20 = size*0x400 + colmode*0x100 + hilo*0x10 + clk;
    if (((hl==0) && (scanmode==1)) || ((hl>=1) && (scanmode==0)))
        ir20 += 4;
    if ((hl>=1) && (scanmode==1))
        ir20 += 8;
    hres = (ir[3]-ir[2])*8;
    vres = ir[7]-ir[6];
    if (scanmode == 1)
        vres *= 2;
    if (scanmode == 2)
        vres /= 2;
}

void dsp_regs()
{
    printf("¥n");
    printf("R0  = %5d ($%04X)¥n",ir[0],ir[0]);
    printf("R1  = %5d ($%04X)¥n",ir[1],ir[1]);
    printf("R2  = %5d ($%04X)¥n",ir[2],ir[2]);
    printf("R3  = %5d ($%04X)¥n",ir[3],ir[3]);
    printf("R4  = %5d ($%04X)¥n",ir[4],ir[4]);
    printf("R5  = %5d ($%04X)¥n",ir[5],ir[5]);
    printf("R6  = %5d ($%04X)¥n",ir[6],ir[6]);
    printf("R7  = %5d ($%04X)¥n",ir[7],ir[7]);
    printf("R20 = %5d ($%04X)¥n",ir20,ir20);
    printf("¥n表示ドット数［横］＊［縦］＝ [%4d]*[%4d]¥n¥n",hres,vres);
```

```
}

void set_regs()
{
    unsigned int    i;
    unsigned short  r20;
    stack = SUPER(0);
    r20 = *(crtc+20) & 0x13;    /* Bit 4,1,0                 */
    if (ir20 >= r20) {        /* 高い画面モードへの切り替え    */
        for (i=0; i<8; i++)
            *(crtc+i) = ir[i];
        *(crtc+20) = ir20;
    }
    else {                  /* 低い画面モードへの切り替え    */
        *(crtc+20) = ir20;
        for (i=1; i<8; i++)
            *(crtc+i) = ir[i];
        *crtc = ir[0];
    }
    if (hl == 2)
        *sysp |=  0x2;
    else    *sysp &= ~0x2;
    *vcon = (ir20 >> 8) & 0x7;
    *(scon+1) = ir[2]+4;
    for (i=0; i<0x200; i++)      /* 時間稼ぎ            */
        ;
    if ((ir20 & 0x1f) == 0)
        *scon = ir[0];
    else    *scon = 0xff;
    *(scon+2) = ir[6];
    *(scon+3) = ir20 & 0xff;
    SUPER(stack);
}
```

4・1 プログラムの実行

プログラムを起動すると，まず，ドットクロックをLow，High，Middleのどれにするかを聞いてきます。Lowは低解像度モード用のクロック群，Highは高解像度モード(69MHz)用クロック群，Middleはシステムポートの HRLビットを‘1’にしたときのクロック群の選択になります。

次に聞いてくるのが分周比です。CRTCのR20のHDビットへの設定値になります。横256ドット，512ドット，768ドット用の分周比のほか，Specialという名前で50MHzのクロックも使えるようにしています。

3番目のパラメータは水平周波数の選択です。公開されている31KHz，15KHzモードのほ

か，VGA互換モードや非公開IOCSにある24KHzモード，640×480ドットモード用の31KHzモードも追加しておきました。

4番目は垂直方向の動作モードです。インターレース，ノン・インターレース，二度読みの3種類が選択できます。

5，6番目はグラフィックの実画面サイズと色モードです。表示ドット数の制御には直接関係ありませんが，R20の設定値を算出するうえで必要なので加えてあります。

なお，すでに述べたとおり，インターレースモードでは垂直同期の周期はレジスタの設定値に0.5を加えた値になりますが，プログラムではこの補正を省略し，計算を簡略化しています。

プログラムを起動し，必要なパラメータを入力すると，それに応じたレジスタの値と表示ドット数を算出して表示します。最後に，この算出値を実際にレジスタに設定するか否かを聞いてきますので，設定したい場合は2を，そのまま終了したい場合は1を入力してください。

設定はCRTCだけでなく，ビデオコントローラやスプライトコントローラに対しても行う必要があります。自分でプログラムを組んで設定するときには，これらへの設定も忘れないようにしてください

4・2 特殊画面モード

最後に，このプログラムの助けを借りながら見つけた画面モードのうち，便利そうなものを2つほど紹介しておきましょう。

1)680×424ドットモード

```
ドットクロック  (1:Low    2:High      3:Middle                    ) 2
ドットクロック  (1:256    2:512       3:768          4:special    ) 2
水平周波数 (1:31KHz 2:31KHz(V) 3:31KHz(VGA) 4:24KHz 5:15KHz       ) 4
走査モード (1:ノン・インターレース  2:インターレース  3:二度読み ) 1
実画面サイズ (1:512*512ドット 2:1024*1024ドット                   ) 2
色数         (1:16色          2:256色     3:65535色               ) 1
R0  =   117 ($0075)
R1  =    10 ($000A)
R2  =    19 ($0013)
R3  =   104 ($0068)
R4  =   464 ($01D0)
R5  =     7 ($0007)
R6  =    32 ($0020)
R7  =   456 ($01C8)
R20 =  1045 ($0415)
```

表示ドット数［横］×［縦］＝ ［680］×［424］

680×424ドットという値は，国産のパソコンでよく使われてきた画面モードである640×400ドットに近く，また水平周波数，縦横のドット数の比率もほぼ同一です。768×512ドットモードのなかに640×400ドットの表示をするよりも大きく表示されますので，特に画像データのやりとりをするような場合には便利なモードでしょう。

2)384×256ドット

```
ドットクロック　(1:Low　 2:High　　3:Middle　　　　　　　　　　　) 3
ドットクロック　(1:256　 2:512　　 3:768　　　　4:special　　　　) 2
水平周波数 (1:31KHz 2:31KHz(V) 3:31KHz(VGA) 4:24KHz 5:15KHz　　 ) 1
走査モード (1:ノン・インターレース　2:インターレース　3:二度読み ) 3
実画面サイズ (1:512 * 512ドット 2:1024 * 1024ドット　　　　　　 ) 1
色数　　　　(1:16色　　　　2:256色　　3:65535色　　　　　　　　 ) 3
R0  =   69 ($0045)
R1  =    7 ($0007)
R2  =   12 ($000C)
R3  =   59 ($003B)
R4  =  567 ($0237)
R5  =    5 ($0005)
R6  =   40 ($0028)
R7  =  552 ($0228)
R20 =  785 ($0311)
```

表示ドット数［横］×［縦］＝ ［376］×［256］

ここでR1を6($0006)，R2を11($000B)に変更して表示領域を８ドット増やすと，横が376＋８＝384になります。384×256ドットという値はゲームを作るのには便利なモードでしょう。縦横の比率は768×512ドットモードと同じで，画面の縦横比とドット数の比がほぼ等しくなります。通常の512×512ドットや256×256ドットではスプライトが横長の長方形になってしまいますが，この画面モードでは正方形で表示されますし，画面も狭いのでキャラクタや背景の高速な回転が要求されるアクションゲームなどを作るときには扱いやすいモードでしょう。

注：正規の方法ではないので，このプログラムではサポートしませんでしたが，水平／垂直の同期期間を正規の値の２倍（水平62KHz，垂直120Hz）にすると，X68000のディスプレイは同期信号を１つおきに取り込んで追従しようとするらしく，ちょうど画面が４分割されたような表示になります。効果的な使い道はあまり思いつきませんが，試してみるのもおもしろいでしょう。

5 ラスター割り込み

ラスター割り込みは，垂直同期の後の水平同期パルスの発生数をカウントしていき，CRTCのR09に書き込まれた値と一致したときにCPUに割り込みをかける機能です。ラスター割り込み要求信号は，MFP（マルチファンクションペリフェラル）の汎用入力端子の1つであるGPIP6に接続されており，CPUに対してはMFP経由で割り込みがかかります。通常，この信号は'1'になっており，一致すると'0'になり，次のラスターになると再び'1'になります。

X68000のラスター割り込み機構の動作は，画面モードがノン・インターレース（高解像度モード，および標準解像度で縦256ドットモードのとき）であるか，インターレース（低解像度の縦512ドットモード）であるかによって若干動作が異なります。

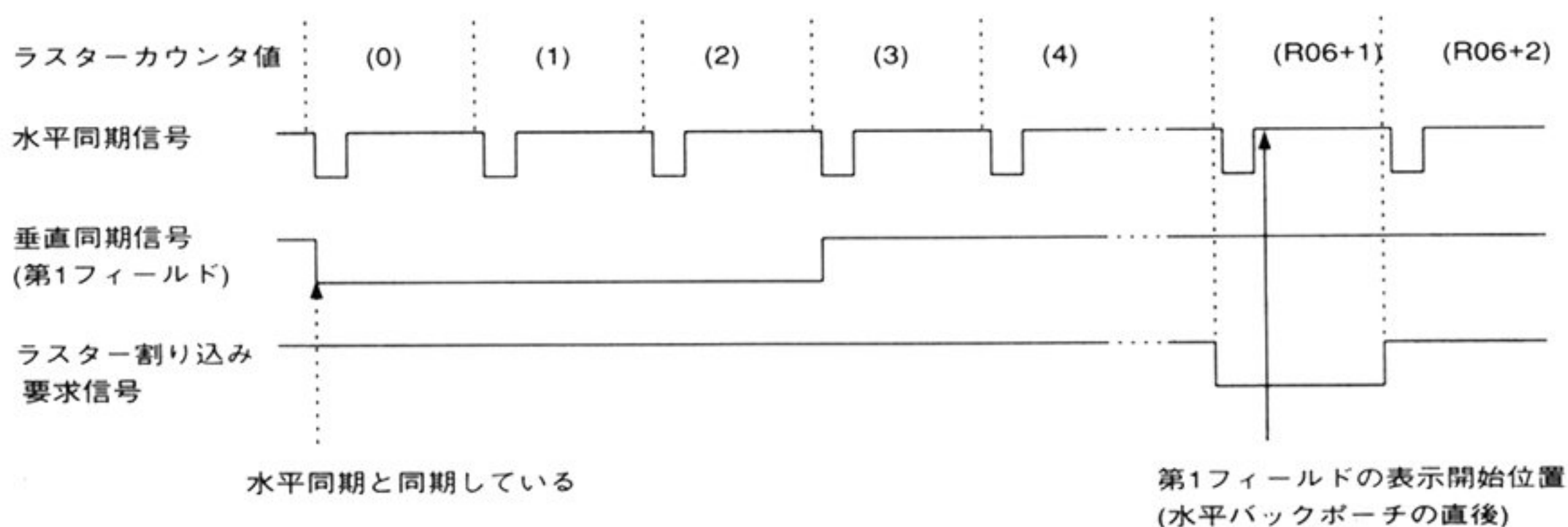
まず，一般的によく使われるノン・インターレースモードのときの動作を説明した後で，インターレースモード時の動作について説明します。

5・1 ノン・インターレースモード時のラスター割り込み動作

ラスターのカウント動作を行う部分を，仮に「ラスターカウンタ」と呼ぶことにします。ラスターカウンタは，垂直同期信号直前の水平同期信号の立ち下がりの少し前に0になり，その後，水平同期信号の立ち下がりがくるたびにその少し前に1ずつ増えていきます。ノン・インターレースモードでは垂直同期の立ち下がりと水平同期の立ち下がりが一致していますが，CRTCは，この最初の水平同期の立ち下がりはカウンタを0にするパルスとしており，次の水平同期信号の立ち下がりからカウントアップしています（図10）。

このカウント値とCRTCのR09に書き込んだ値が一致すると，割り込み要求信号が'0'になり，不一致だと'1'になります。このため，割り込み要求信号が'0'になっている期間は水平同期期間と同じ時間になります。

ラスターカウンタのカウント動作が垂直同期信号の立ち下がりから行われるため，R09に書き込むラスター番号と画面上に表示されるラスターの番号は一致していません。つまり，画面上の一番上のラスターの開始位置で割り込みをかけるためには，((垂直同期パルス幅)＋(垂直バックポーチ))÷(水平同期期間)の値をR09に設定することになります。これは，CRTCのレジスタR06への設定値に1を加えたものと同じです。

●図10……第1フィールドの水平・垂直同期信号の波形とラスターカウンタの値

＊垂直同期パルス幅を3（R05の値を2）に設定した場合の波形を示す。
＊ノン・インターレースモード時の波形も同様。

たとえば，31KHzモードの標準的な画面モードではR06は$28が設定値ですから，R09には$29（10進数で41）を設定すると，画面の最上部の水平同期期間の開始位置で割り込みがかかるわけです。

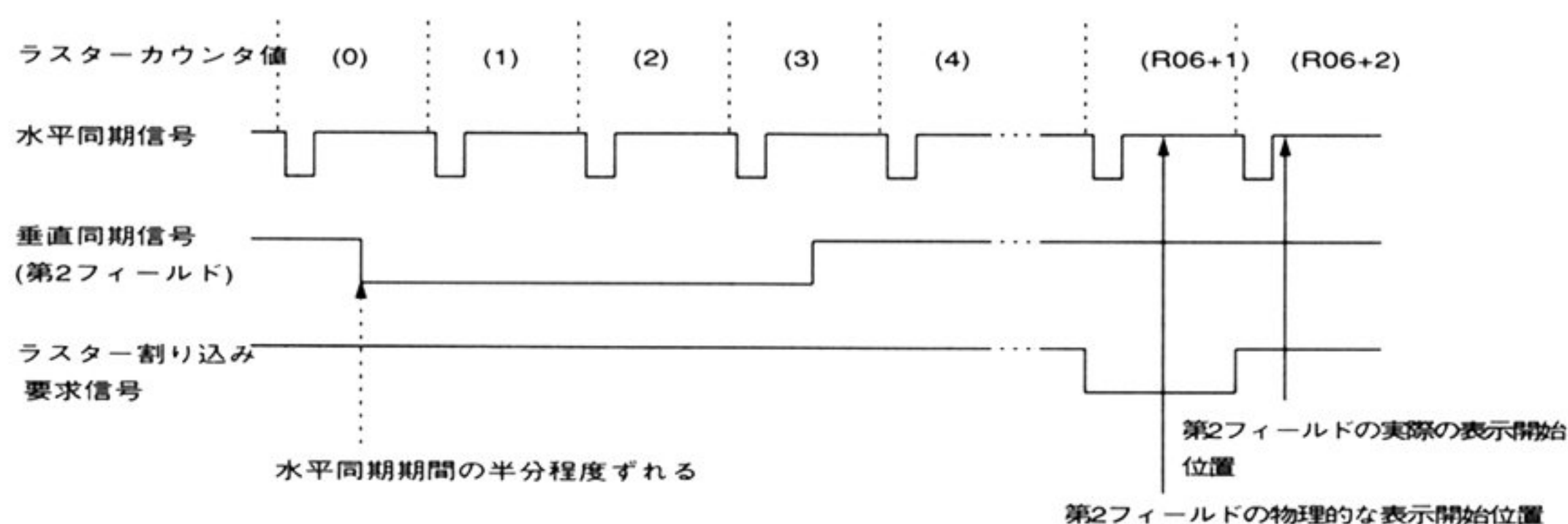
一般に画面上のN本目（一番上は０とする）のラスターの開始位置で割り込みをかけるには，(R06+1+N)を設定すればよいことになります。ラスター割り込み要求信号は，GPIPの状態データレジスタ($E88001)のビット６で読み出すこともできます。

ラスター割り込みのなかで表示開始位置を変更する（ラスタースクロール）場合には，スクロールを開始させたいラスターの１本前で割り込みをかけるように設定します。スクロールしたいラスターの先頭で割り込みをかけても，CRTCはすでに表示開始アドレスを取り込んでしまっているため，CPUがレジスタを書き換えても，その値が有効になるのは次のラスターからになってしまうためです。実際に画面の上からN本目のラスターをスクロールさせるには，R09に(R06+N)の値を設定すればよいことになります。

5・2 インターレースモード時のラスター割り込み動作

次にインターレースモード時のラスター割り込みの動作について説明します。インターレースモードのときは１枚の画面（フレーム）を第１フィールド，第２フィールドの計２回に分けて表示しています。このため，インターレースモードでのラスター割り込み動作はノン・インターレースモードのときとは異なっています。

インターレースモードのときもノン・インターレースのときと同じような動作を期待するのであれば，上から偶数本目（０，２，４……）のラスターを指定したのであれば，第１フィー

●図11……第2フィールドの水平・垂直同期信号の波形とラスターカウンタの値

＊垂直同期パルス幅を3(R05の値を2)に設定した場合の波形を示す。

ルドのときに割り込みが発生し，奇数本目のラスターを指定したときには第2フィールドのときだけ割り込みが発生しなければなりません。また，インターレースモードでは1本おきに走査しているわけですから，ラスター番号の数え方も表示期間中は2ずつ増やさないと画面上の番号と一致しなくなります。

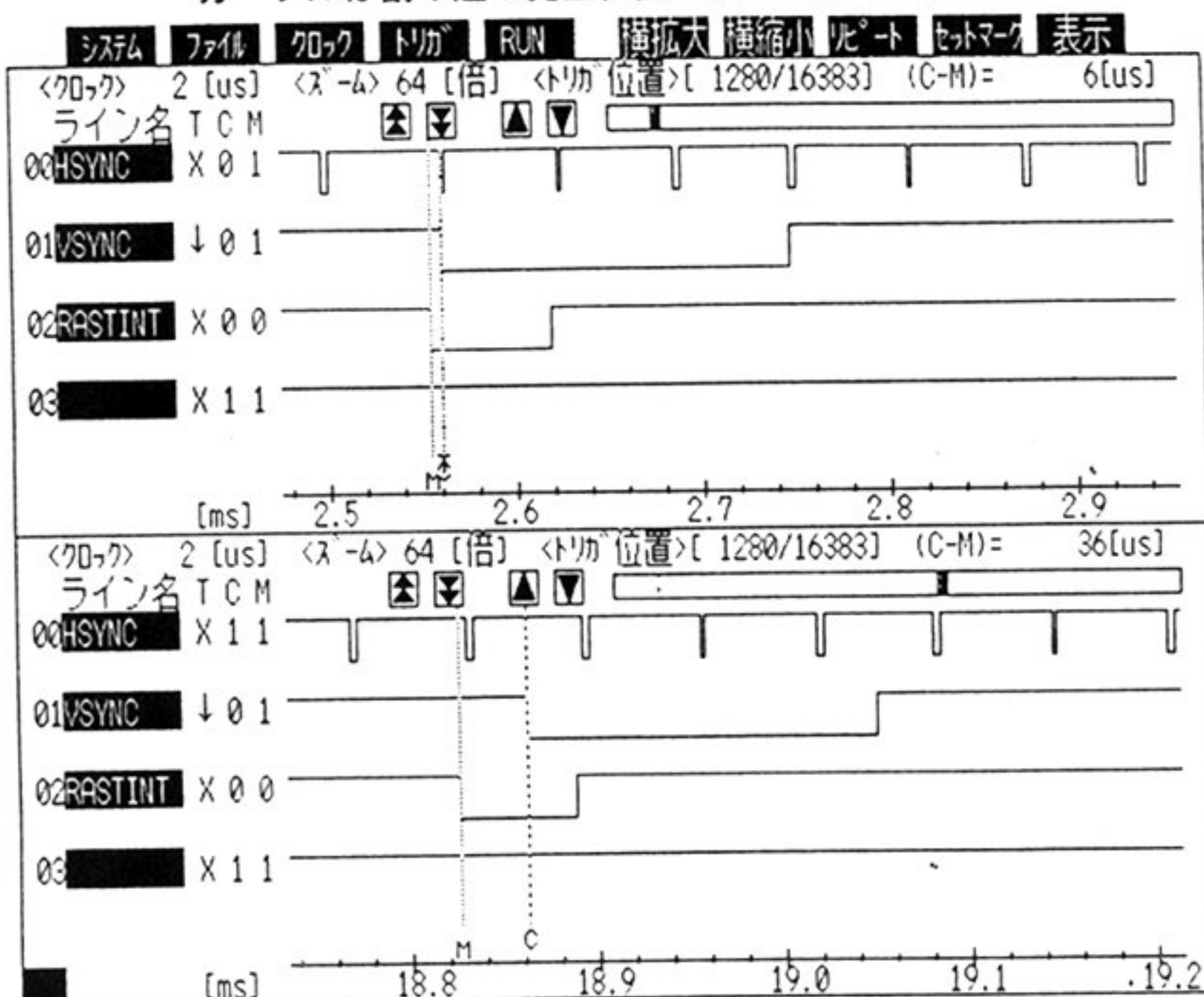
ところが，実際にはラスターカウンタのカウント方法は，インターレースモードのときもノン・インターレースモードのときと同じように垂直同期の直前の水平同期信号の立ち下がりの少し前で0になり，その後，水平同期の立ち下がりがくるたびに1ずつ増えていき，R09と一致すると割り込み要求信号が‘0’になるという動作をします(図11)。つまり，X68000のラスター割り込み機構では，インターレースモードにすると偶数本目のラスターでも奇数本目のラスターでも割り込みが発生してしまうわけです。

このため，割り込みの発生位置は，画面の最上部のラスターを0番目とすると(R09－R06)×2，および(R09－R06)×2－1本目のラスターの開始位置ということになります。

5・3 実際のラスター割り込み信号の波形

図12に，インターレースモード時の水平同期信号と垂直同期信号，およびラスター割り込み信号の波形を示します。図の上半分が第1フィールドのときの波形，下半分が第2フィールドの波形を示しています。ノン・インターレースモードのときは第1フィールドと同様の波形になります。

いずれの場合も，ラスター割り込み信号は水平同期信号の立ち下がりの少し前（ほぼ8ドットクロックの3.5倍程度前）に変化していることがわかります。

●図12……実際のラスター割り込み波型
上半分が第一フィールド，下半分が第二フィールド。
カーソルは割り込み発生位置と水平同期の立ち下がり

6 スーパーインポーズ／水平同期アジャスト

6・1 スーパーインポーズ動作の概略

X68000は，専用ディスプレイ上でTVやビデオ画面とコンピュータ画面を重ね合わせて表示するスーパーインポーズ機能を持っています。スーパーインポーズ動作時には，ディスプレイは，TVやビデオ画像を表示しながらコンピュータ側から出力されてくる映像信号を合成して表示します。表示のタイミングの元となる水平同期や垂直同期信号はTVやビデオ入力のものが使われ，コンピュータ画面のほうの同期信号は無視されます。

つまり，通常はディスプレイ側がコンピュータ側のタイミングに追従してくれるのですが，スーパーインポーズ動作のときにはコンピュータ側がディスプレイの表示タイミングにあわせて映像信号を送り出さなければならないわけです。このため，X68000の専用ディスプレイ制御

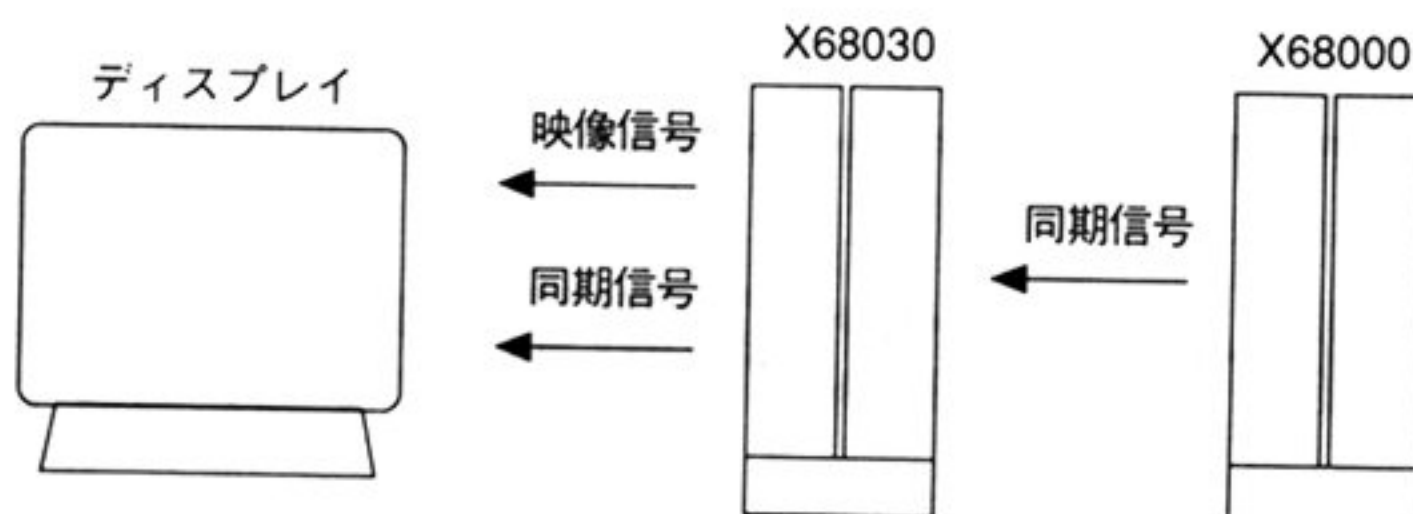
コネクタにはディスプレイ側からの同期信号入力端子（1，2番ピン）が設けられています。

スーパーインポーズ動作時にはもう1つ気をつけなければならないことがあります。ディスプレイはTVやビデオに同期しているため，コンピュータ側の表示開始タイミングによってはコンピュータ画面が画面の左右にずれてしまうのです。これを調整するために設けられたのがCRTCのR08です。

このレジスタの設定と，X68000のスーパーインポーズ動作の関係を調べてみました。

6・2 外部同期信号への追従

スーパーインポーズ動作時の波形を見るため，X68000から取り出した同期信号をX68030の同期信号入力に接続し，X68030の同期信号出力がどのように動作するかを観察してみました(図13)。信号極性は，いろいろ試した結果，水平同期を正極性，垂直同期を負極性とするのが最も自然な動きのように思えましたので，そのようにしてあります。

●図13……実験時の接続

この状態でX68000とX68030の動作をいろいろ変えてみると，次のようなことがわかりました。

1)スーパーインポーズ動作をしようとするのは15KHzモード（39MHzのクロックを使う画面モード）のときだけで，31KHzモードでは行われない。
2)外部からの同期信号入力があると，スーパーインポーズ動作をしようとする
3)同期信号入力に追従できないときは無視して動作する

このうち，2)と3)については少し説明が必要でしょう。

6・2・1 CRTCによる同期動作

スーパーインポーズ動作時の外部同期信号との同期動作はCRTCによって行われますが，スーパーインポーズ動作の動作ON/OFFはキーボードコントローラによって制御されています。キーボードの［SHIFT］+［+］でスーパーインポーズ動作のトグルが行われますが，このことはホストであるX68030側は関知していません。実際，X68030のI/Oポートを眺めていてもスーパーインポーズ動作の許可／禁止を制御するようなビットは見あたりません。

では，CRTCのほうはどのようにして外部からの同期信号に追従するか否かを決めているのでしょうか。これは，X68000とX68030のケーブルを接続していてわかりました。CRTCは，基本的に外部からの同期信号入力があると，その信号に追従して動作しようとし，同期信号入力がなくなるなど追従できない状態になると，CRTC自身のタイミングで動作し始めるのです。

つまり，［SHIFT］+［+］によるスーパーインポーズ動作のON/OFFというのは，ディスプレイ側に対してコントロール信号端子に同期信号を出力するか否かを指示していたというわけです。

6・2・2 インターレース／ノン・インターレース

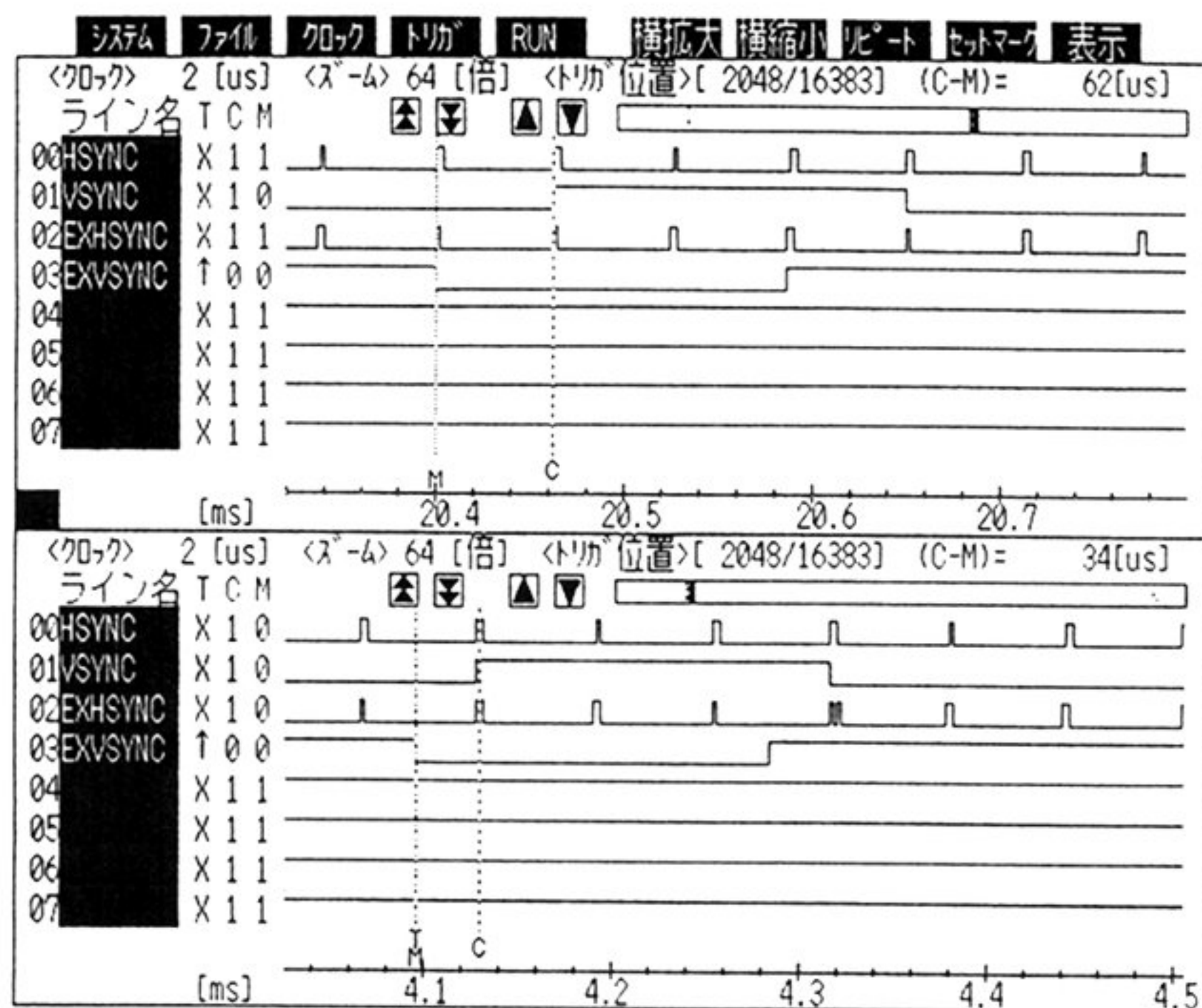
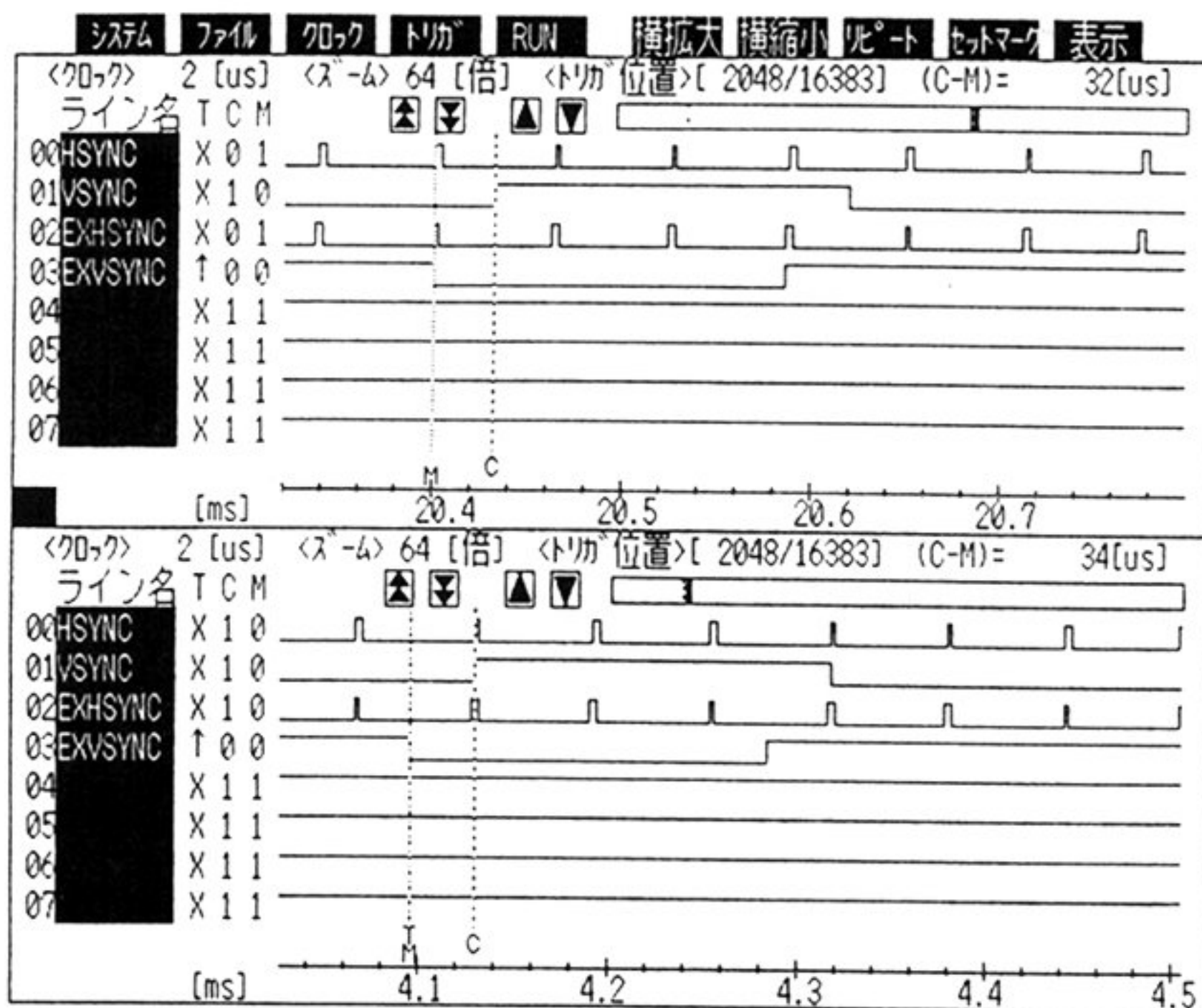
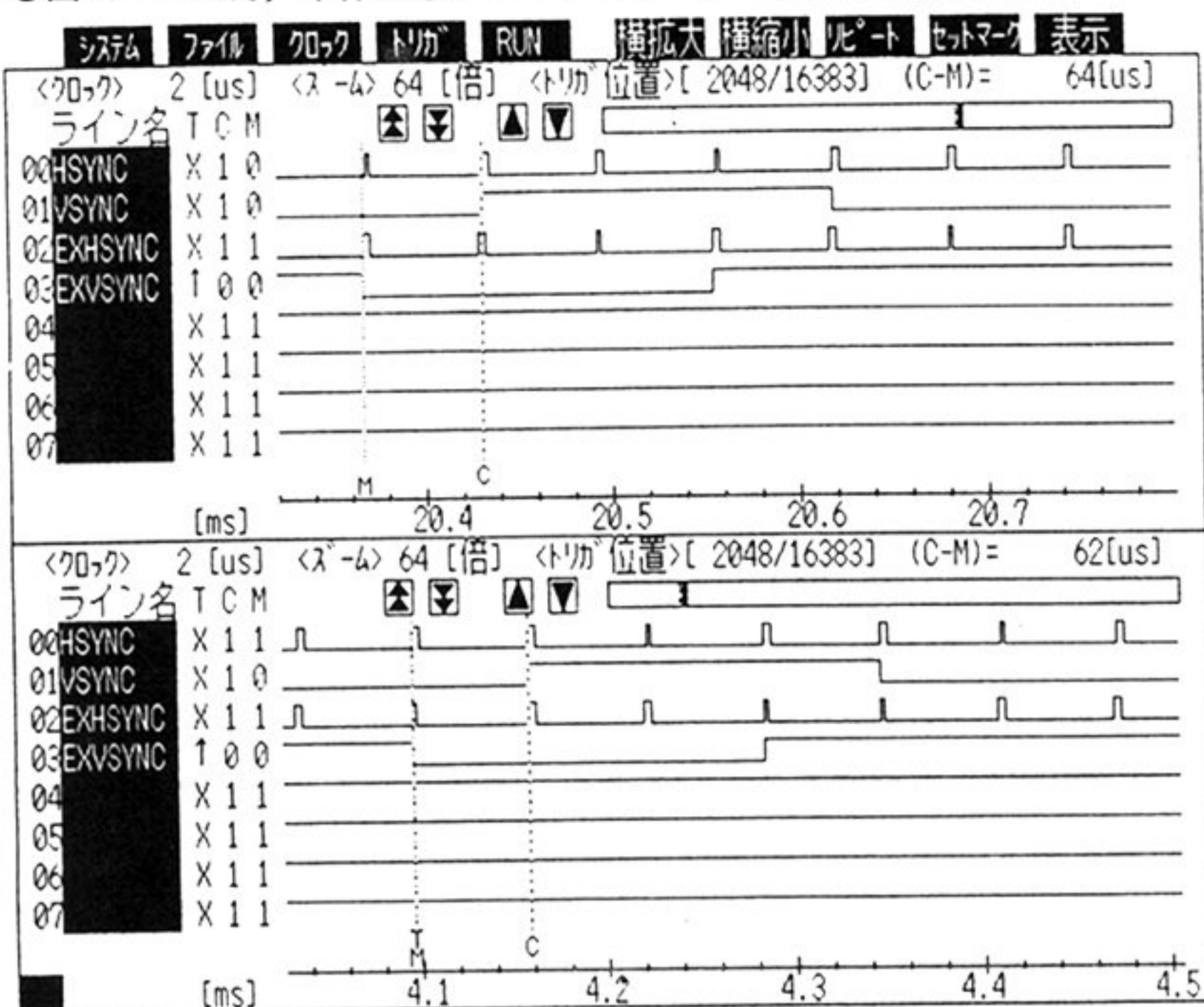
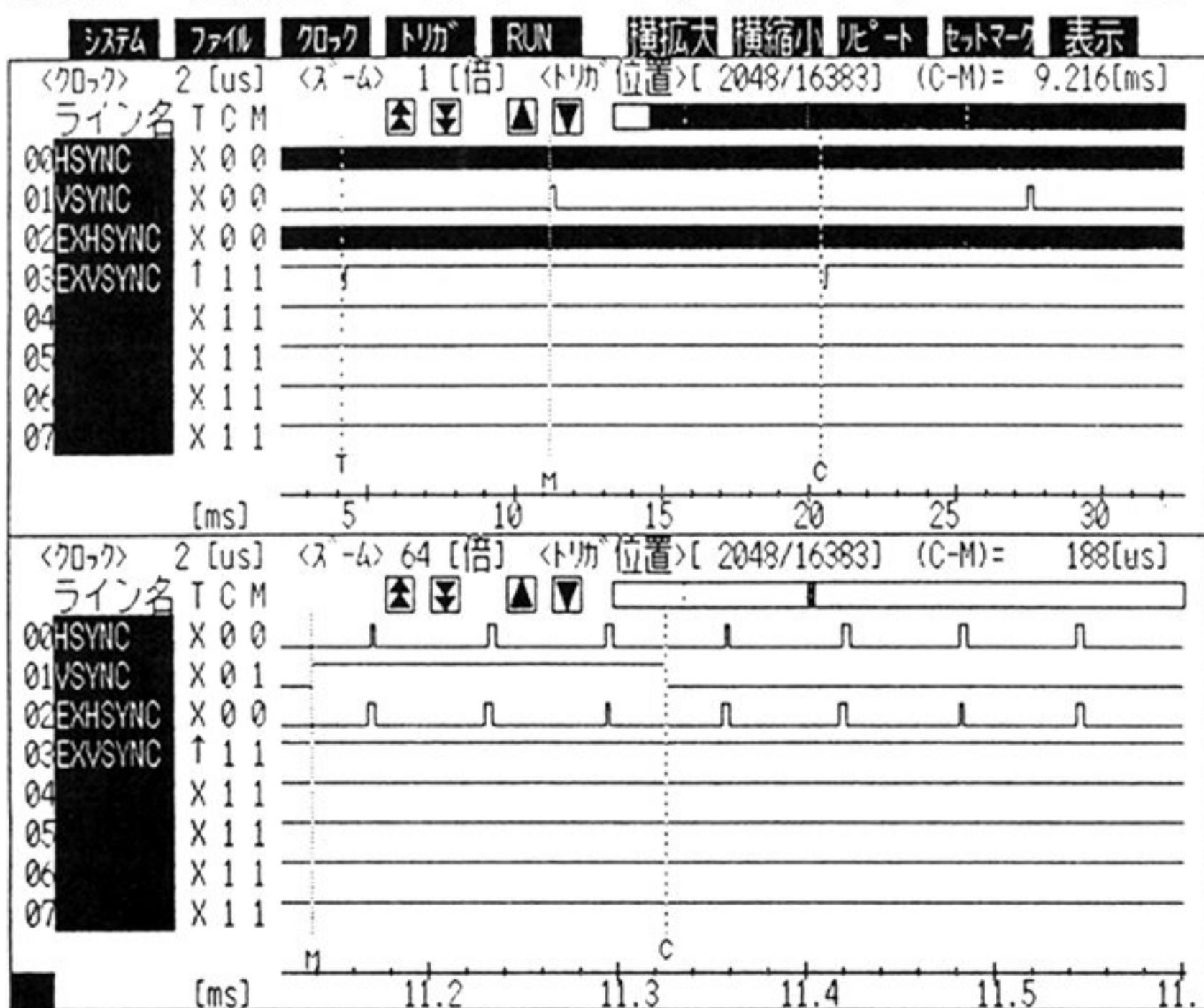
TVやビデオの信号はインターレース動作しかありませんが，実験ではX68000と接続しているのでノン・インターレースモードの同期信号を与えることもできます。

ここで，X68000とX68030をそれぞれインターレースモード（縦512ドットモード），ノン・インターレースモード（縦256ドットモード）に切り替えながら組み合わせたときの動作を見てみました。

図14～図17にそれぞれの動作波形を示します。図14は入力信号がインターレース，X68030がノン・インターレースのときの波形，図15は両者ともインターレース，図16は両者ともノン・インターレース，図17は入力信号がノン・インターレースで，X68030がインターレースモードのときの波形です。

図15を見ると，入力信号が第１フィールドのときに本体は第２フィールド用の映像信号を，入力信号が第２フィールドのときには第１フィールドの映像信号を出力しているらしいことがわかります。

また，図14と図15から，X68030がノン・インターレースモードのときには入力がインターレース／ノン・インターレースのいずれであっても同期がかかり，ノン・インターレース動作を

●図14……入力がインターレース，本体がノン・インターレースのときの動作

●図15……入力，本体ともインターレースのときの動作

●図16……入力，本体ともノン・インターレースのときの動作

●図17……入力がノン・インターレース，本体はインターレースのときの動作

●図18……外部同期信号と本体の走査モードによる動作の変化

入力	本 体	
	インターレース	ノン・インターレース
インターレース	インターレースで同期 (フィールドは反転)	ノン・インターレースで同期
ノン・インターレース	同期不可 (同期入力がないのと同じ)	ノン・インターレースで同期

することがわかります。

図17はほかの図とは少し違っていますが，これはX68030側が同期しなかったので，拡大率を変更したためです。図では表現できませんが，測定していると入力された同期信号とまったく関係なくX68030側が動作していることがわかります。これは，X68030が第２フィールドに同期しようとしているにもかかわらず，外部からは第１フィールドのタイミングの同期信号しか入ってこないため，CRTCは同期信号に追従することができず，同期信号入力がない場合と同じ扱いになっていると考えられます。

走査モードの組み合わせと，そのときの動作を図18にまとめておきますので参考にしてください。

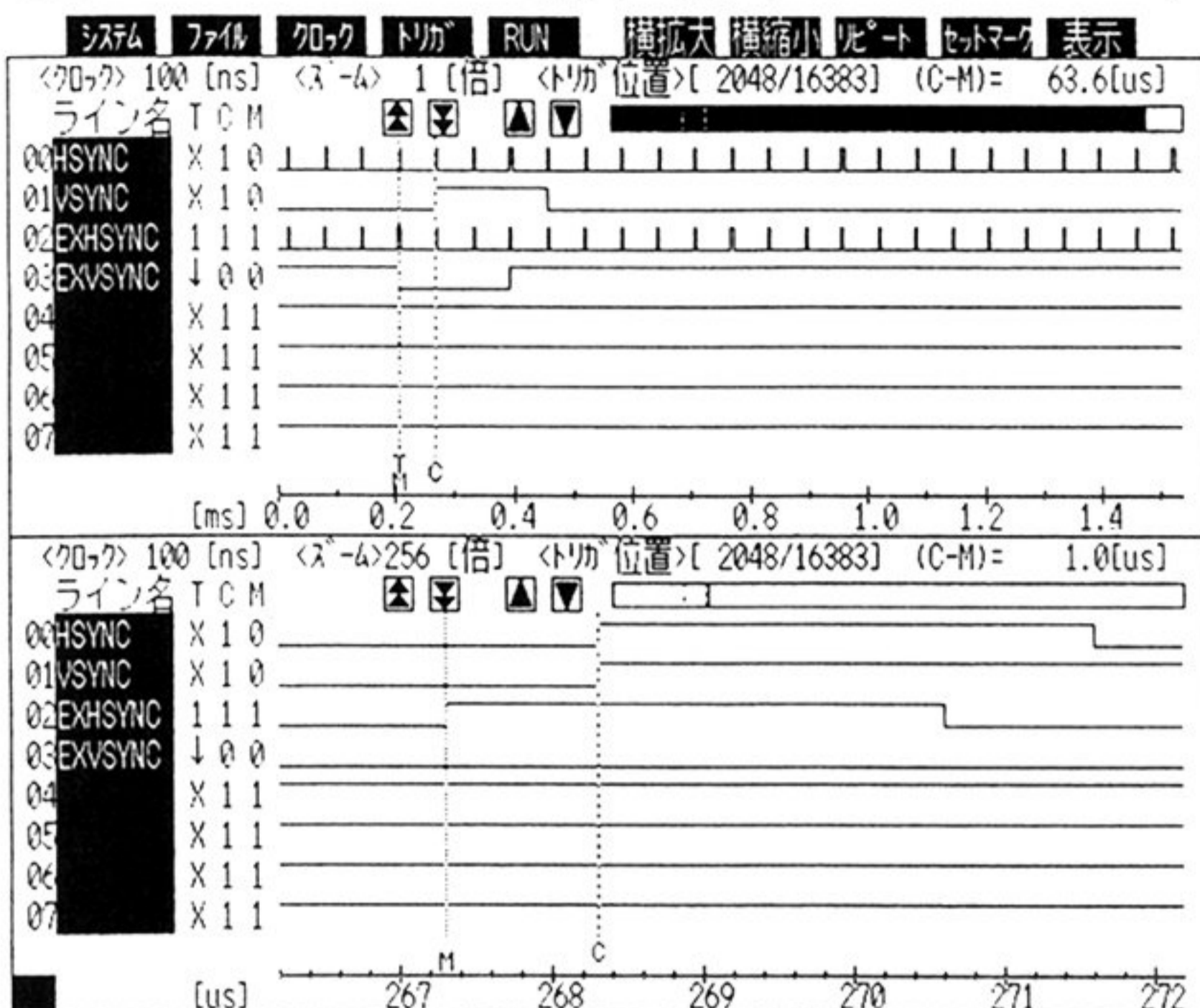
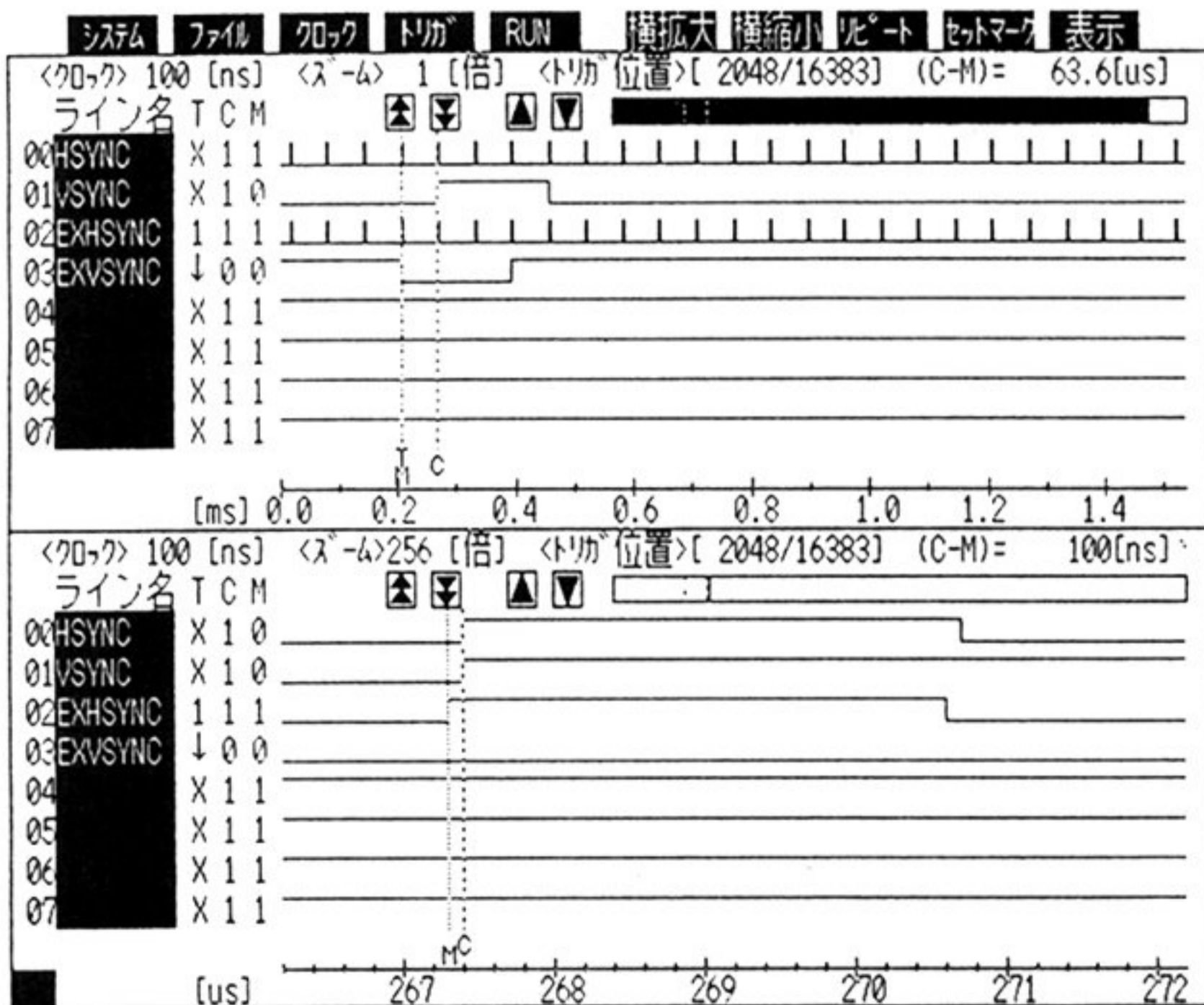
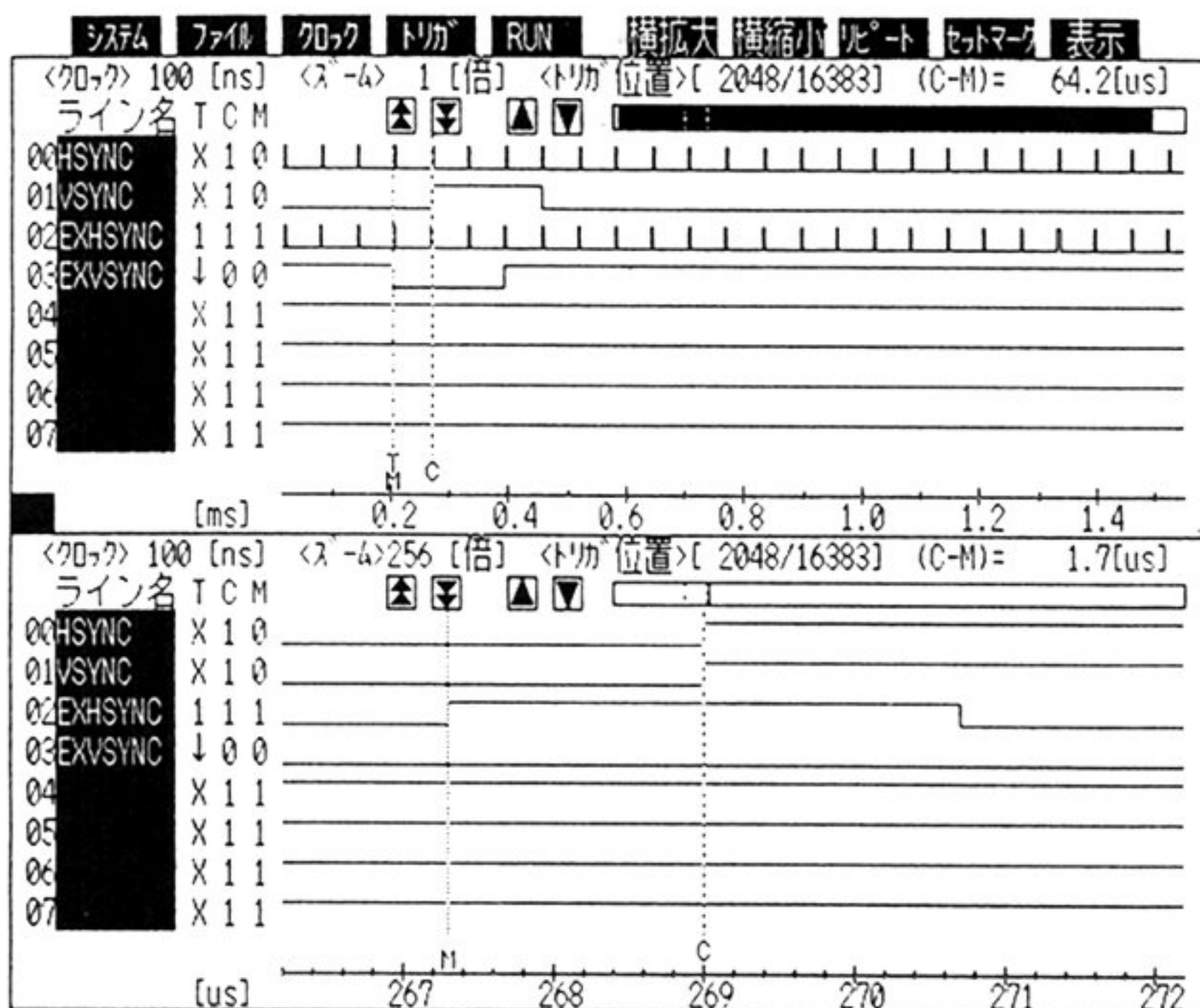
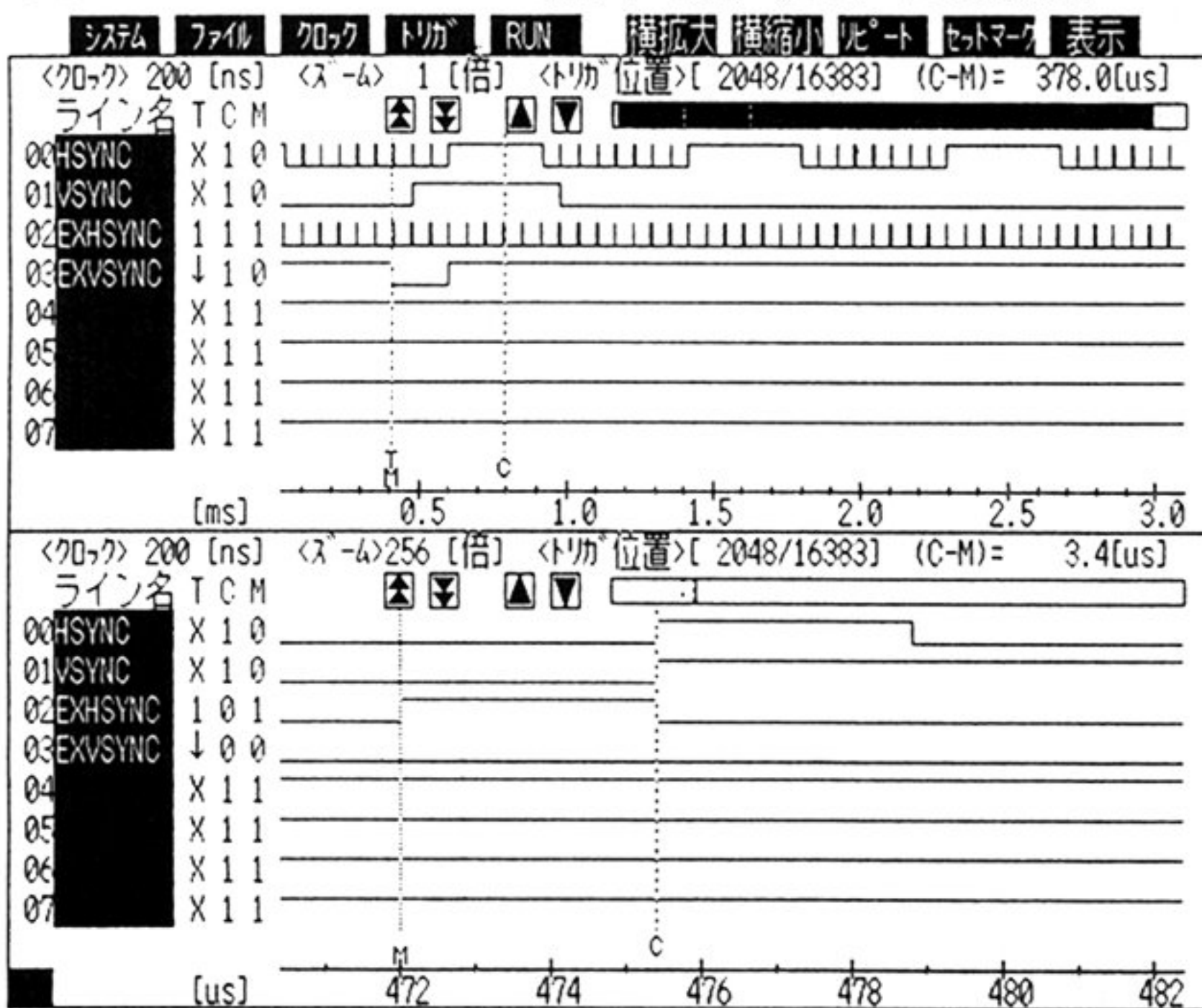
6・3 水平同期アジャスト

外部の水平同期信号と本体のディスプレイタイミングの微調整を行うために設けられているのが，CRTCのR08です。X68000とX68030を両方とも256×256ドットモードにしておいてR08の設定値を変更したときの波形を，図19〜22に示します。図19は画面モード３を設定したままの状態（$24が設定されているはず），図20〜22はそれぞれR08の設定値を$00，$40，$81にしたときの波形です。

図19〜21を見ると，R08の値が増えると，外部水平同期信号の立ち上がりから本体の水平同期信号出力の立ち上がりまでの時間が長くなっていくことがわかります。

もう少し詳しく調べてみると，この時間はR08への設定値が１増加するごとに約25nS程度増加します。これをCRTCに関係するクロックと照らし合わせてみると，ちょうどドットクロックとして使われている38.86363MHzの１サイクル分（約25.731nS）にあたります。CRTCは，このクロックで外部からの同期信号をサンプリングして動作しているものと思われます。

設定を０にしたときの遅れ時間は走査モードによって変化します。何度か測定してみると，ノン・インターレース（256×256ドット）時には３クロック分，インターレース（512×512ドット）時には４クロック分になっているようです。ここからR08への設定値をNとしたときの水平同期入力と水平同期出力の間の時間は，ノン・インターレースモードのときは(3+N)÷

●図19……スーパーインポーズ動作時の波形（256×256ドットモード，デフォルト値）

●図20……スーパーインポーズ動作時の波形（R08に＄00を設定）

●図21……スーパーインポーズ動作時の波形（R08に＄40を設定）

●図22……スーパーインポーズ動作時の波形（R08に＄81を設定）

38.86363, インターレースモードのときは(4+N)÷38.86363(単位はいずれもμS)となります。

さらにR08の設定値を増やしていくと，あるところから急に動作がおかしくなります。徐々に値を増やしていってみると，$81以上で同期動作が異常になります。このときの波形が図22です。CRTCの出力する同期信号の波形がかなりおかしなものになっていることがわかります。図22の下半分に示すとおり波形を拡大してみると，R08の設定による遅れ時間が，入力される同期信号のパルス幅程度まで大きくなってきたときからCRTCの同期動作がおかしくなるようです。

X68000の15KHzモード時の水平同期パルス幅は，計算上約3.2936μSです。これをクロック数に直すと，3.2936×38.86363より128(=$80)となります。この値が限界値を示しているように思われます(ちなみに，$81を設定したときの遅れ時間は計算上約3.3965μSとなります。図22から実測値でも3.4μとなっており，計算値と一致していることが読み取れます)。

また，R08の値が多少大きい程度なら，CRTCは図22のように変な波形を出力しながら同期をかけようとしますが，極端に大きくなると同期信号の出力も行わなくなり，動作を停止させてしまいます。こうなると，CPUからVRAMへのアクセスもできなくなり，ハングアップしてしまいます。R08の設定を変更するときには注意してください。

数値演算プロセッサ

数値演算プロセッサは浮動小数点演算を高速に処理するLSIです。ここでは，X68030に搭載された数値演算プロセッサとX68000シリーズに搭載されたものとの違いと，数値演算プロセッサの使い方を説明します。

1 X68000との違い

CPUと密着してあたかもCPUの一部となったかのように動作し，CPUの機能を拡張するLSIのことを「コプロセッサ」と呼びます。X68000やX68030で使われている数値演算プロセッサMC68881やMC68882は本来コプロセッサとして設計されたLSIで，CPUとの連係動作をさせるのが本来の姿なのですが，X68000シリーズではCPU自身にコプロセッサと接続する機能がなかったため，やむなくI/Oデバイスの1つとして接続していました。このため，X68000シリーズで数値演算プロセッサを扱うときは，煩雑なコプロセッサとのやりとりをすべてプログラムでコントロールしなければなりませんでした。

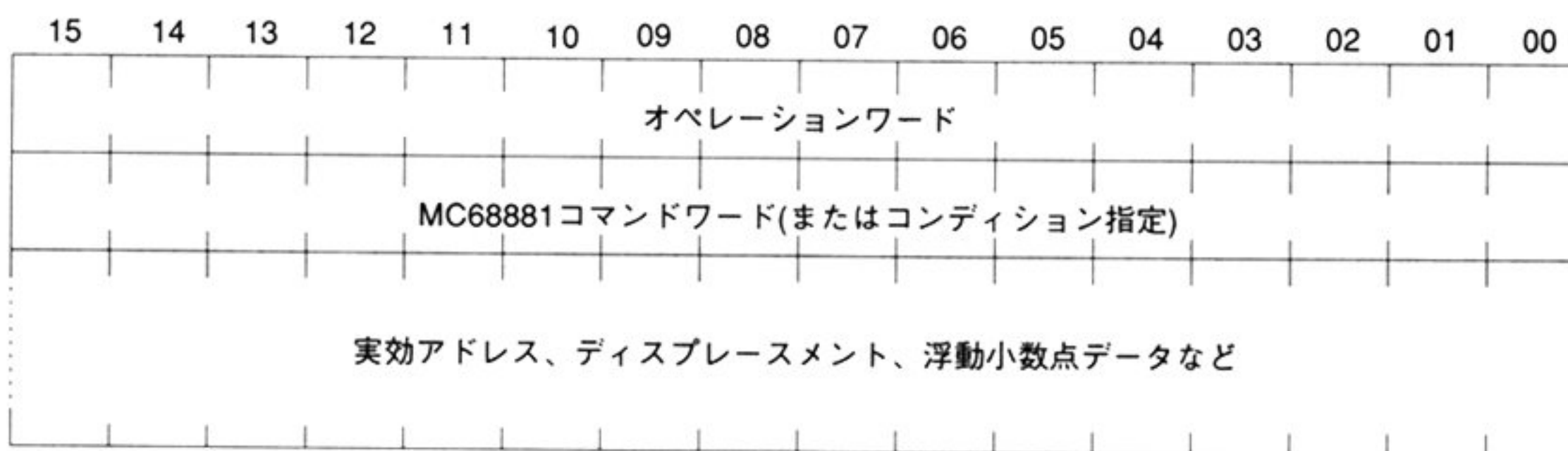
一方，X68030のCPU，MC68EC030にはCPU自体にコプロセッサと直接コミュニケーションする機能が備わっているため，数値演算プロセッサはCPUと直結され，本来のコプロセッサとして動作させることが可能になりました。コプロセッサとの面倒なやりとりはすべてCPUが自動的に実行するようになり，浮動小数点演算命令を並べるだけで演算が行えます。

2 数値演算命令

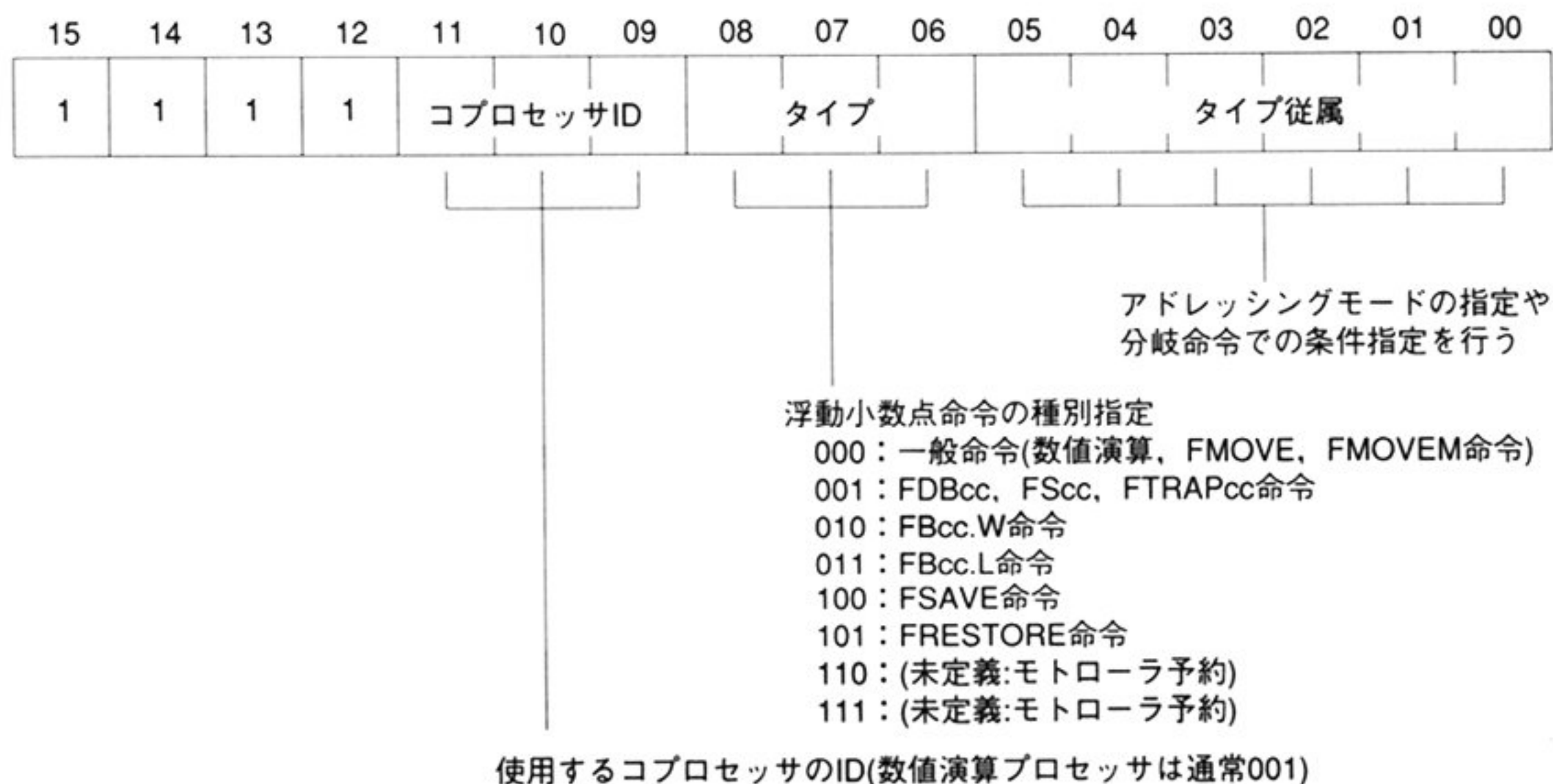
2・1 数値演算命令のフォーマット

数値演算プロセッサ用の命令のフォーマットを図1に示します。最初のワードは「オペレーションワード」と呼ばれ，アドレッシングモードの指定，コプロセッサと連係してどのような動作を行うのかなどを指定するものです。次にくる1ワードのデータは「コマンドワード」（またはコンディション）と呼ばれ，コプロセッサに対する命令にあたります。必要ならばその後に，浮動小数点データなどのオペランドが並びます。

実は2ワード目のデータは，X68000で数値演算プロセッサを扱うとき，最初に書き込んでいたコマンドそのものです。つまり，最初のワードがCPUに対してコプロセッサとの連係動作の開始を指示し，2ワード目でCPUがコプロセッサに与えるべきコマンドを指定していたわけです。パラメータが必要であれば，1ワード目でパラメータの取り込み方（すなわちアドレッシング方法）が指定されているので，CPUはそれに従ってデータの入出力を実行するというわけです。

●図1……浮動小数点命令のフォーマット

2・2 オペレーションワードの構造

浮動小数点演算命令の先頭ワードである，オペレーションワードのビット配置は図2のよう

●図2……オペレーションワードのビット配置

になっています。コプロセッサIDというのは，コプロセッサの番号を示すものです。MC68020以降のCPUでは最大8個までのコプロセッサを接続することができるようになっています。それぞれのコプロセッサはIDと呼ばれる固有の番号を持っており，オペレーションワードのコプロセッサIDフィールドでどれを使うかを指定します。モトローラでは，数値演算プロセッサのIDは通常‘001’を使用することにしているため，X68030でも‘001’で使用するようになっていま

C O L U M N

コプロセッサらしく

アセンブラが浮動小数点命令に対応していれば，このような命令パターンの展開はすべてアセンブラが行ってくれますので，プログラマがビットの並びなどを気にする必要はありません。たとえば，浮動小数点レジスタの0番にA0が指している先のデータ（単精度実数とする）をロードし，A1が指しているデータを掛けるというときには，

```
FMOVE.S (A0),FP0
FMUL.S  (A1),FP0
```

などと書けばよいわけです。これは，通常のMOVEやMUL命令の頭にFをつけ，型指定がBやWなどのかわりにS(単精度実数)，レジスタ名称が浮動小数点レジスタを示すFP0になっただけのものに見えます。まるで浮動小数点命令と浮動小数点レジスタが増えたかのように見えることがわかるでしょう。

す。IDの値はハードウェアによって決まりますので，ソフトウェアでIDを変更することはできません。

ビット6～8のタイプフィールドは数値演算プロセッサ命令のタイプを指定します。通常の演算命令はすべて‘000’，FSAVEとFRESTORE命令はそれぞれ‘100’と‘101’になります。タイプフィールドが‘001’，‘010’，‘011’の命令は浮動小数点演算のステータスを使った分岐命令です。数値演算プロセッサの演算ステータスが使われるほかは，CPU自身の命令である分岐命令と同様です。

ビット0～5のタイプ従属フィールドは，アドレッシングモードやコンディションを指定するビットです。このフィールドの使われ方はタイプフィールドの値によって変わります。

2·3 タイプ000（一般命令）の命令フォーマット

オペレーションワードのタイプフィールドが‘000’のときの命令フォーマットを図3に示します。通常使用する浮動小数点演算命令やデータ転送命令などはすべてこのタイプになります。

●図3……タイプ000の命令フォーマット

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
1	1	1	1	コプロセッサID			0	0	0	MODE			REGISTER		
Opclass			RX			RY			EXTENSION						

2·3·1 オペレーションワード

オペレーションワードのタイプ従属ビット（下位6ビット）でアドレッシングモードを指定します。タイプ従属ビットのうち，ビット3～5を「MODEフィールド」，0～2を「REGISTERフィールド」と呼びます。MODEフィールドとREGISTERフィールドで指定されるアドレッシングモードを図4に示します。アドレッシングモードの処理はCPUが行うので，基本的にはCPUが持っているアドレッシングモードはすべて利用できます。ただし，アドレスレジスタと数値演算プロセッサの間のデータ転送というのはほとんど意味がないため，アドレスレジスタ直接のモードだけは省略されています。

●図4……アドレッシングモードの指定

アドレッシングモード	MODE	REGISTER
Dn	000	アドレスレジスタ番号
An		
(An)	010	アドレスレジスタ番号
(An)+	011	アドレスレジスタ番号
-(An)	100	アドレスレジスタ番号
(d16,An)	101	アドレスレジスタ番号
(d8,An,Xn)	110	アドレスレジスタ番号
(bd,An,Xn)	110	アドレスレジスタ番号
([bd,An,Xn],od)	110	アドレスレジスタ番号
([bd,An],Xn,od)	110	アドレスレジスタ番号
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d16,PC)	111	010
(d8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

拡張ワード(省略フォーマット)

15	14–12	11	10–09	08	07–00
D/A	レジスタ	W/L	スケール	0	ディスプレースメント

拡張ワード(完全フォーマット)

D/A：インデックスレジスタはデータレジスタ(0)/アドレスレジスタ(1)
W/L：インデックスサイズは符号拡張ワード(0)/ロングワード(1)
スケール：スケールファクタ(00：×1 01：×2 10：×4 11：×8)
BS：ベースレジスタを加算する(0)/ベースレジスタをサプレスする(1)
IS：インデックスオペランドを加算する(0)/インデックスオペランドをサプレスする(1)
BDサイズ：ベースディスプレースメントのサイズ(00：未使用 01：ヌル 10：ワード 11：ロングワード)
I/IS：インデックス/間接の選択

IS	I/IS	アドレッシング
0	000	メモリ間接なし
0	001	ヌルアウタディスプレースメント、プリインデックス付き間接
0	010	ワードアウタディスプレースメント、プリインデックス付き間接
0	011	ロングアウタディスプレースメント、プリインデックス付き間接
0	100	モトローラ予約
0	101	ヌルアウタディスプレースメント、ポストインデックス付き間接
0	110	ワードアウタディスプレースメント、ポストインデックス付き間接
0	111	ロングアウタディスプレースメント、ポストインデックス付き間接
1	000	メモリ間接なし
1	001	ヌルアウタディスプレースメント付きメモリ間接
1	010	ワードアウタディスプレースメント付きメモリ間接
1	011	ロングアウタディスプレースメント付きメモリ間接
1	1XX	モトローラ予約

MODEフィールド（ビット3～5）が‘111’以外のときにはREGISTERフィールドで使用するレジスタの番号を指定します。たとえば，MODEフィールドが‘000’でREGISTERが‘001’なら，D1との間でデータの転送を行うことを意味するわけです。

MODEフィールドが‘111’のときには，さらにREGISTERフィールドによってアドレッシングモードの選択が行われます。

表を見ると，REGISTERフィールドとMODEフィールドが同一のものがあります。これらの区別はさらにコマンドワードの後につく拡張ワードで行われます。拡張ワードのフォーマットと各ビットの意味も図4に付加しておきましたので参考にしてください。これらの拡張ワードは，MC68EC030の通常の命令のアドレッシングモードのものとまったく同じです。

コマンドワードのビット14（「R/Mビット」と呼びます）が‘0’の場合，すなわち，浮動小数点レジスタしか使わない命令の場合にはREGISTERやMODEフィールドによるアドレッシングモードの指定は無効です。コマンドワードのR/Mビットが‘0’のときには，MODEフィールドとREGISTERフィールドはすべて‘0’にしておいてください。

2・3・2 コマンドワード

コマンドワードは，クラス，RX，RY，拡張の4つのフィールドに分けられます。命令タイプと各フィールドの関係を図5に示します。コマンドワードの内容の詳細については『Inside X68000』の数値演算プロセッサの章の7・1～7・3を参照してください。

アドレッシングモードにイミディエイトを指定した場合には，コマンドワードの後にデータをそのまま置きます。

●図5……コマンドワードの各フィールドの値と命令の関係

Opclass	RX	RY	EXTENSION	命令タイプ
000	FPn(ソース)	FPn(ディスティネーション)	命令指定 (FMOVE, FADD等)	浮動小数点レジスタ間転送／演算命令
001				未定義(モトローラ予約)
010	データフォーマット (000～110)	FPn(ディスティネーション)	命令指定 (FMOVE, FADD等)	外部から浮動小数点レジスタへの転送／演算
010	111	FPn(ディスティネーション)	ROMのオフセットアドレス	FMOVECR命令
011	データフォーマット	FPn(ソース)	パックトBCD(RX=011)のときはKファクタ，それ以外のときは0	浮動小数点レジスタから外部への転送
100	FPcrの選択	000	0	浮動小数点コントロールレジスタへの書き込み
101	FPcrの選択	000	0	浮動小数点コントロールレジスタの読み出し
110	レジスタリストの指定方法と転送順序	00X	転送するレジスタの選択(マスクパタン/データレジスタ番号)マスクパタンとして使う場合はビット数が不足するのでRYビットの最下位も利用する	FMOVEM命令(外部から浮動小数点レジスタへの転送)
111		00X		FMOVEM命令(浮動小数点レジスタから外部への転送)

2・4 タイプ001（FDBcc/FScc/FTRAPcc）の命令フォーマット

オペレーションワードのタイプフィールドが‘001’のときの命令フォーマットを図6に示します。このタイプに入る命令には，FDBcc/FScc/FTRAPcc（ccには判定条件が入ります。たとえば，FTRAPEQなら，比較の結果が等しければトラップ例外が発生します）があります。これらの命令は，それぞれCPUが持っている命令であるDBcc, Scc, TRAPcc命令の数値演算プロセッサ版に相当します。条件判定が数値演算プロセッサで行われるほかはCPUの持つ命令と同じような動作になります。

●図6……タイプ001の命令フォーマット

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
1	1	1	1	コプロセッサID			0	0	1	命令定義					
0	0	0	0	0	0	0	0	0	0	コンディション					

2・4・1 オペレーションワード

オペレーションワードの下位6ビットは命令定義フィールドです。このフィールドの値と各命令の関係は図7のようになっています。これらの命令のフォーマットは基本的にFScc命令のものですので，これをタイプ000のときのようにMODEとREGISTERに分けてみました。タイプ000のオペレーションワードと比較すると，FDBccやFTARPcc命令はこのビットパターンのうち，未使用のものやアドレッシングモードの一部（プログラムカウンタを使った間接アドレッシング）をつぶして割り当てていることがわかります。

●図7……命令定義フィールドと命令の関係

命令定義フィールド		命令
(MODE)	(REGISTER)	
000	レジスタ番号	FScc Dn
001	レジスタ番号	FDBcc Dn,<label>
010	レジスタ番号	FScc (An)
011	レジスタ番号	FScc (An)+
100	レジスタ番号	FScc −(An)
101	レジスタ番号	FScc d16(An)
110	レジスタ番号	FScc (d8,An,Xn) FScc (bd,An,Xn) FScc ([bd,An,Xn],od) FScc ([bd,An],Xn,od)
111	000	未定義(モトローラ予約)
111	001	未定義(モトローラ予約)
111	010	FTRAPcc.W #<data>
111	011	FTRAPcc.L #<data>
111	100	FTRAPcc (パラメータなし)
111	101	未定義(モトローラ予約)
111	110	未定義(モトローラ予約)
111	111	未定義(モトローラ予約)

2・4・2 コマンドワード

タイプ001の命令では，コマンドワード下位6ビットのコンディションフィールドでどのような条件判定を行うのかを指定します。上位10ビットは無視されていますので0以外になっていてもかまいませんが（Fライン例外などは発生しません），将来使用される可能性もあるので，ここは0にしておくようにしてください。

コンディションフィールドの値とその内容の関係は，『Inside X68000』の数値演算プロセッサの章の7・6を参照してください。

2・5 タイプ010/011（FBcc.W/FBcc.L）の命令フォーマット

オペレーションワードのタイプフィールドが010/011の命令はFBcc命令です。命令フォーマットを図8に示します。条件判定を数値演算プロセッサに行わせるほかはBcc命令と同じです。条件が成立すれば，ディスプレースメントをプログラムカウンタに加算した番地に分岐します。タイプフィールドが010のときには16ビット（ワード）ディスプレースメント，011だと32ビット(ロングワード)のディスプレースメントが使われます。Bccと異なり，8ビットディスプレースメントはありません。

ディスプレースメントは2の補数として扱われます。たとえば，ディスプレースメントが16ビット$FF00ならば，プログラムカウンタの値から$100を引いた番地を指すことになります。

オペレーションワードの下位6ビットで指定する条件判定の内容は，タイプ001命令のコマンドワードの下位6ビットと同じです。

●図8……浮動小数点命令のフォーマット（タイプ010/011）

2・6 タイプ100（FSAVE）/101（FRESTORE）の命令フォーマット

タイプ100はFSAVE命令，タイプ101はFRESTORE命令です。命令フォーマットと使用可能なアドレッシングモードを図9に示します。

FSAVE命令は現在の数値演算プロセッサの動作を停止し，通常プログラマからはアクセスできない内部情報を取り出す命令です。FRESTORE命令はFSAVE命令で取り出された情報を数値演算プロセッサに戻し，中断されたところから演算処理を再開する命令です。

これらの命令はマルチタスクOSの下で複数のタスクが数値演算プロセッサを使う場合には必須となるものです。タスクが切り替わるときにFSAVE命令を使って数値演算プロセッサの内部情報をすべて退避させます。再び同じタスクが起動されるときにFRESTORE命令で前の

●図9……タイプ100/101の命令フォーマットと使用可能なアドレッシングモード

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
1	1	1	1	コプロセッサID			1	0	1/0	MODE			REGISTER		

アドレッシングモード	MODE	REGISTER	備考
Dn			
An			
(An)	010	アドレスレジスタ番号	
(An)+	011	アドレスレジスタ番号	FRESTORE命令(ビット6=1)のみ
-(An)	100	アドレスレジスタ番号	FSAVE命令(ビット6=0)のみ
(d16,An)	101	アドレスレジスタ番号	
(d8,An,Xn)	110	アドレスレジスタ番号	
(bd,An,Xn)	110	アドレスレジスタ番号	
([bd,An,Xn],od)	110	アドレスレジスタ番号	
([bd,An],Xn,od)	110	アドレスレジスタ番号	
(xxx).W			
(xxx).L			
#<data>	111	100	
(d16,PC)	111	010	FRESTORE命令(ビット6=1)のみ
(d8,PC,Xn)	111	011	FRESTORE命令(ビット6=1)のみ
(bd,PC,Xn)	111	011	FRESTORE命令(ビット6=1)のみ
([bd,PC,Xn],od)	111	011	FRESTORE命令(ビット6=1)のみ
([bd,PC],Xn,od)	111	011	FRESTORE命令(ビット6=1)のみ

状態に復帰させることで，複数のタスクで数値演算プロセッサを使うことができるようになるわけです（FP0～FP7，FPCR，FPSR，FPIARの各レジスタについてはFMOVEM命令で別途退避／復帰を行う必要がある）。また，演算エラーなどで例外が発生したときの処理でも，これらの命令が使用されます。

3 MC68881とMC68882の違い

X68030で使用されている数値演算プロセッサMC68882は，X68000用で数値演算プロセッサとして使われていたMC68881の性能向上版にあたり，ソフトウェア的にはコンパチブルということになっていますが，細かく見ていくといくつか変更されている点が見受けられます。

MC68882とMC68881との大きな違いは，

1）MC68882では，演算動作をある程度平行処理できるようにしていること

2）FSAVE/FRESTORE命令で扱われるステートフレームのサイズが変わっていること

3）プロトコルバイオレーション（プロトコル違反）例外の発生条件が変わっていること

の3点です。

3・1 平行動作のサポート

MC68881とMC68882の内部ブロックを図10に示します。MC68881の内部は，大きくBIU (Bus Interface Unit) とAPU (Arithmetic Processing Unit) の2つのユニットで構成されています。BIUがCPUとの間のやりとりを受け持ち，APUが実際の演算処理を実行します。MC68882では，この2つのユニットに加えてCU (Conversion Unit) が追加されています。一見小さな変更ですが，このCUのおかげでMC68882はMC68881に比べ，効率のよい演算処理動作が行えるようになっています。

3・1・1 CUの動作の概要

MC68882で追加されたCUは，BIUのなかにあるオペランドCIRと連結されており，単精度(略号：S)/倍精度(略号：D)/拡張精度(略号：X)の各データフォーマットと，MC68882の内部フォーマットとの変換を行うユニットです。

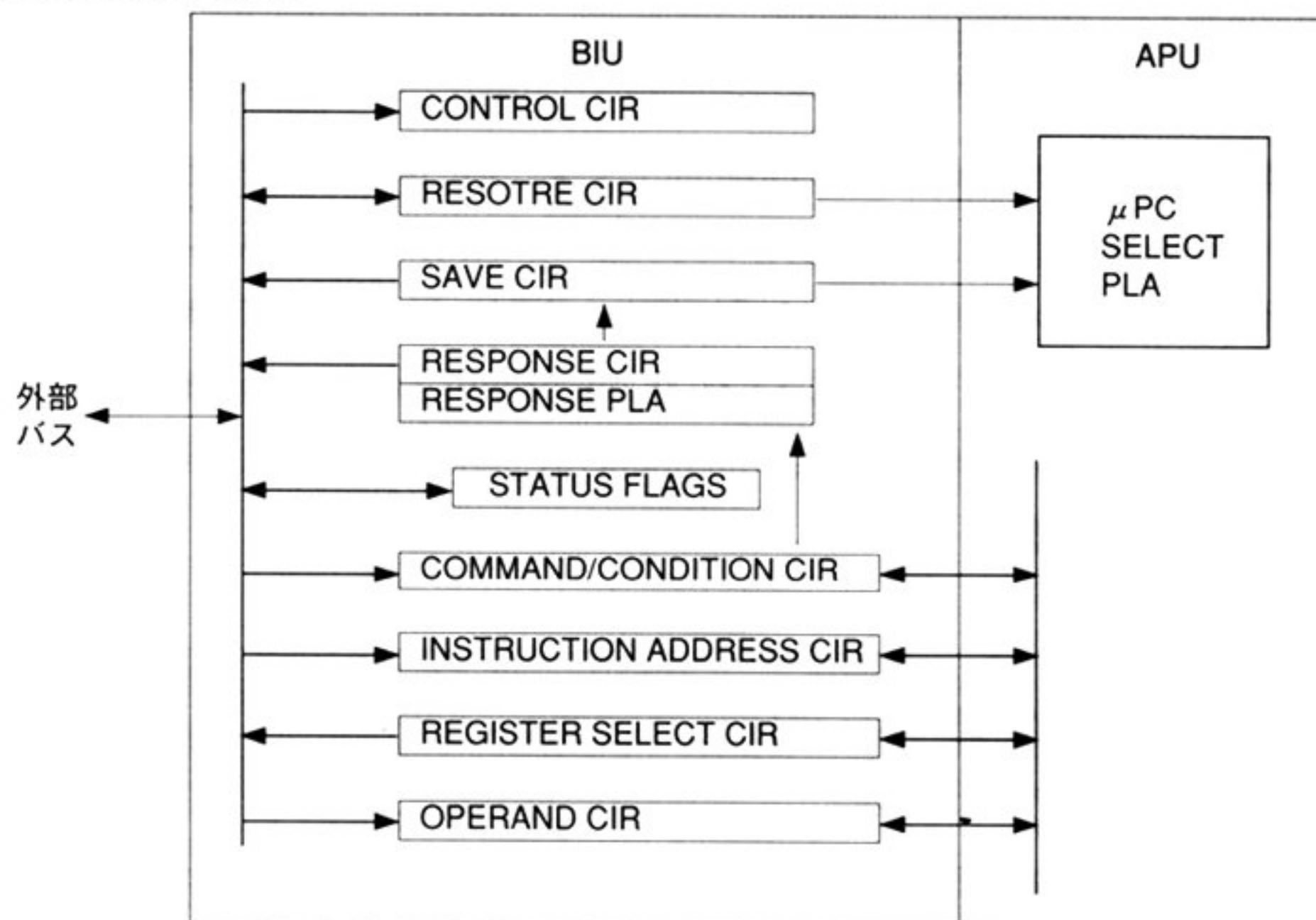
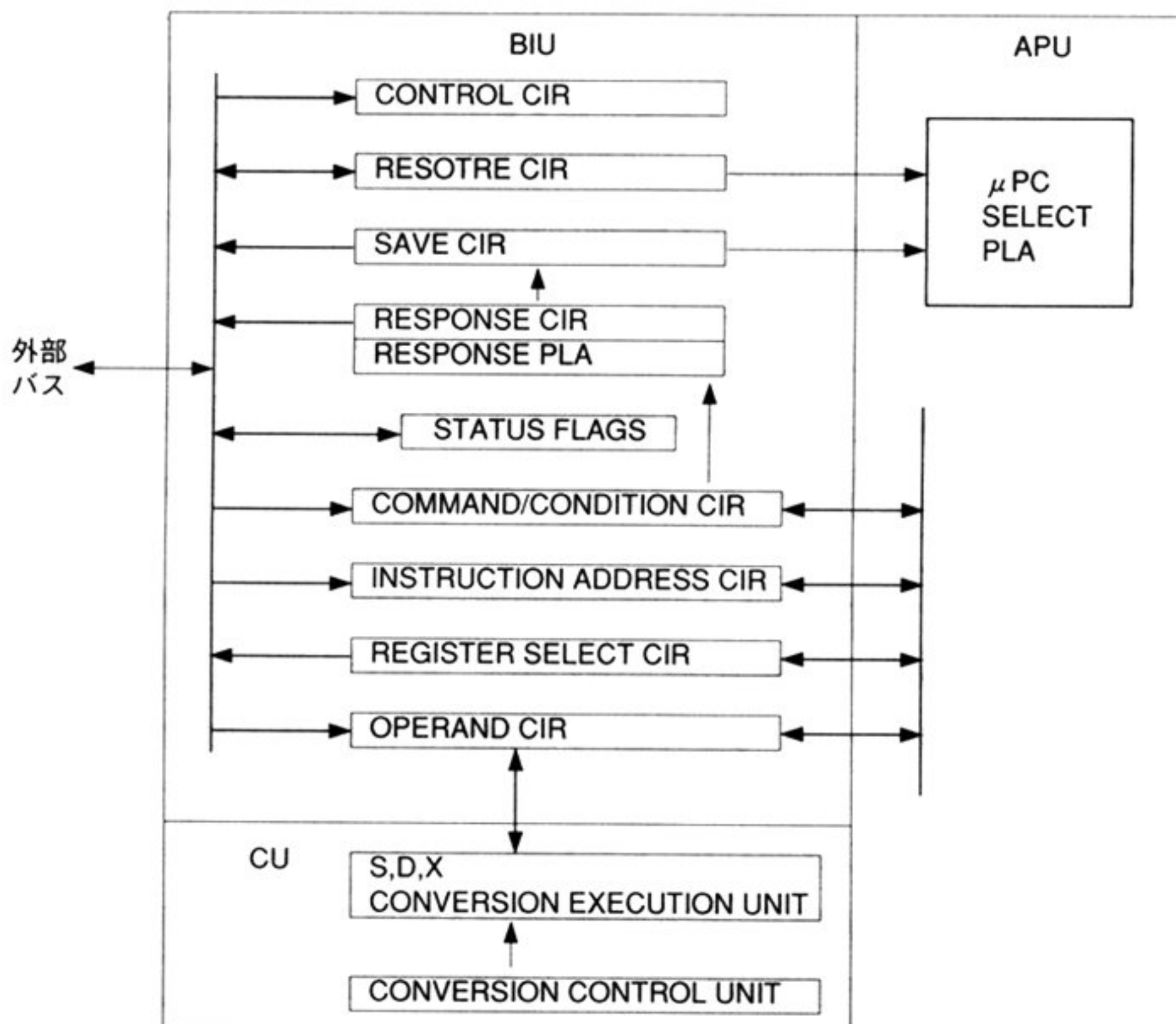
数値演算プロセッサとの間でデータのやりとりをする場合，数値演算プロセッサは内部と外部とのデータフォーマットの変換作業を行う必要があります。MC68881では外部とのフォーマット変換はすべてAPUが処理しているため，オペランドの転送が終わった後，BIUが次の命令を受け付けたとしても，データの転送はAPUが最初の演算処理が完了するまで行えませんでした。

これに対し，MC68882の場合にはCUがAPUから独立しているため，データのフォーマットがS, D, Xのいずれかであれば，APUが演算処理中であってもデータ変換作業を進めることができるようになっています(これ以外のフォーマットとの間の変換は，MC68881と同じようにAPUが処理します)。

CUが特に有効に働くのは，時間のかかる演算命令の次にFMOVE命令がある場合です。FMOVE命令の対象となるレジスタが衝突していない場合，すなわち先に実行されている演算命令のディスティネーションになっていなければ，CUはAPUの動作完了を待つ必要がありませんので，APUの演算処理と平行してデータの変換，転送が実行されます。もちろん，浮動小数点レジスタ間の転送もレジスタの衝突がないかぎり実行されます。

演算命令の次も演算命令であった場合，演算処理自体はAPUで行わざるを得ません。しかし，ソースオペランドがCUで変換可能なフォーマットであれば (FSIN.S #1.9, FP0などの場合)，CUはオペランドの変換作業を行い，APUが処理を完了するのを待ちます。APUがあ

●図10……MC68881とMC68882の内部ブロック

○MC68881内部ブロック

○MC68882内部ブロック

けば，CUはAPUに変換済みのオペランドを引き渡し，また次の変換作業に入ることが可能になります。

このときはFMOVE命令のような完全な平行動作にはなりませんが，データの変換作業時間は事実上なくなったことになりますので，その分，演算処理時間の改善が図られたことになります。

3・1・2 CUによる平行動作の例

CUの効果がどのように現れるのかを，次のようなプログラムを例にとって見ていきましょう。

```
FMUL.S   #1.2,FP0
FMUL.S   #2.3,FP1
FMOVE.S FP0,(A0)
```

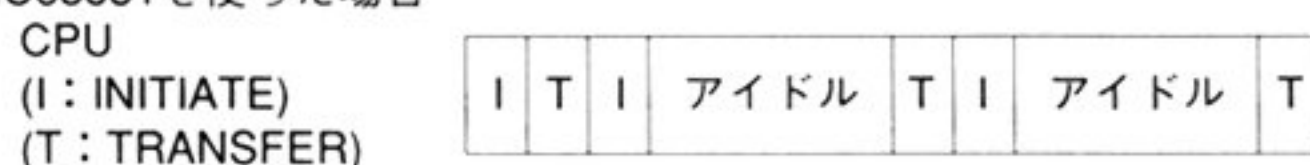
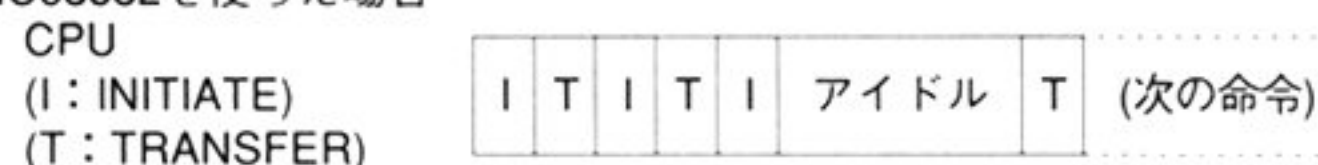
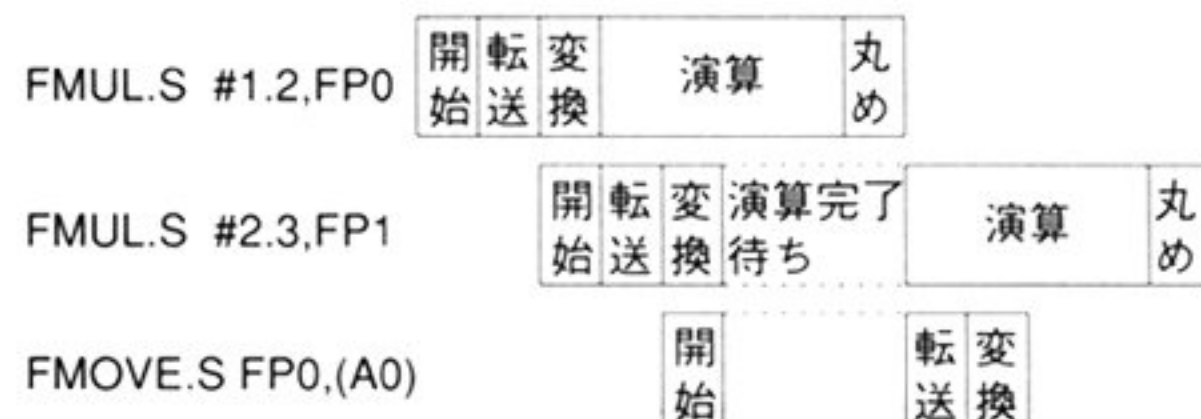
図11にMC68881で処理させた場合の動作とMC68882で処理させた場合の動作を示します。

MC68881の場合，最初のFMULがAPUによる変換作業に入った時点でBIUは次のコマンドを受け付けますが，APUが演算処理中なのでオペランドの転送以降は最初のFMULの完了待ちになります。以後，同じようにデータ変換を含めた処理はすべて前の命令の実行完了まで行えません。

これに対してMC68882の場合，APUが最初のFMULの演算を始めた時点で，CUは次のFMUL命令のオペランド変換作業に移ります。変換が始まれば，BIUは次の命令を受け付けます。

さらに最初のFMUL演算処理が完了した時点で，APUは次のFMULの演算に入るわけですが，この時点で3番目のFMOVE命令が実行可能となります。FMOVE命令で転送されているのがFP0であり，2番目のFMUL命令の演算とは関係がないためです。CUは最初のFMULの実行が完了した時点でFP0の値を引き出し，データ変換をして外部に転送します。

CPUは，この転送が終了した時点で次の命令の実行に取り掛かれますので，プログラムの実行効率も改善されます。

●図11……数値演算プロセッサの動作例

3・2 ステートフレームの違い

MC68881とMC68882のもう1つの大きな違いは，FSAVE命令やFRESTORE命令で扱われるステートフレーム（数値演算プロセッサの内部情報）のサイズが変わったということです。

MC68881とMC68882のステートフレームを図12に示します。MC68881/MC68882のステートフレームは，ヌルステートフレーム，アイドルステートフレーム，ビジーステートフレームの3種類があります。

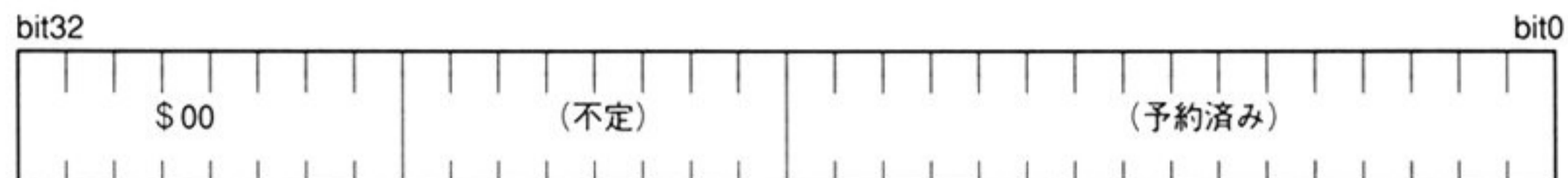
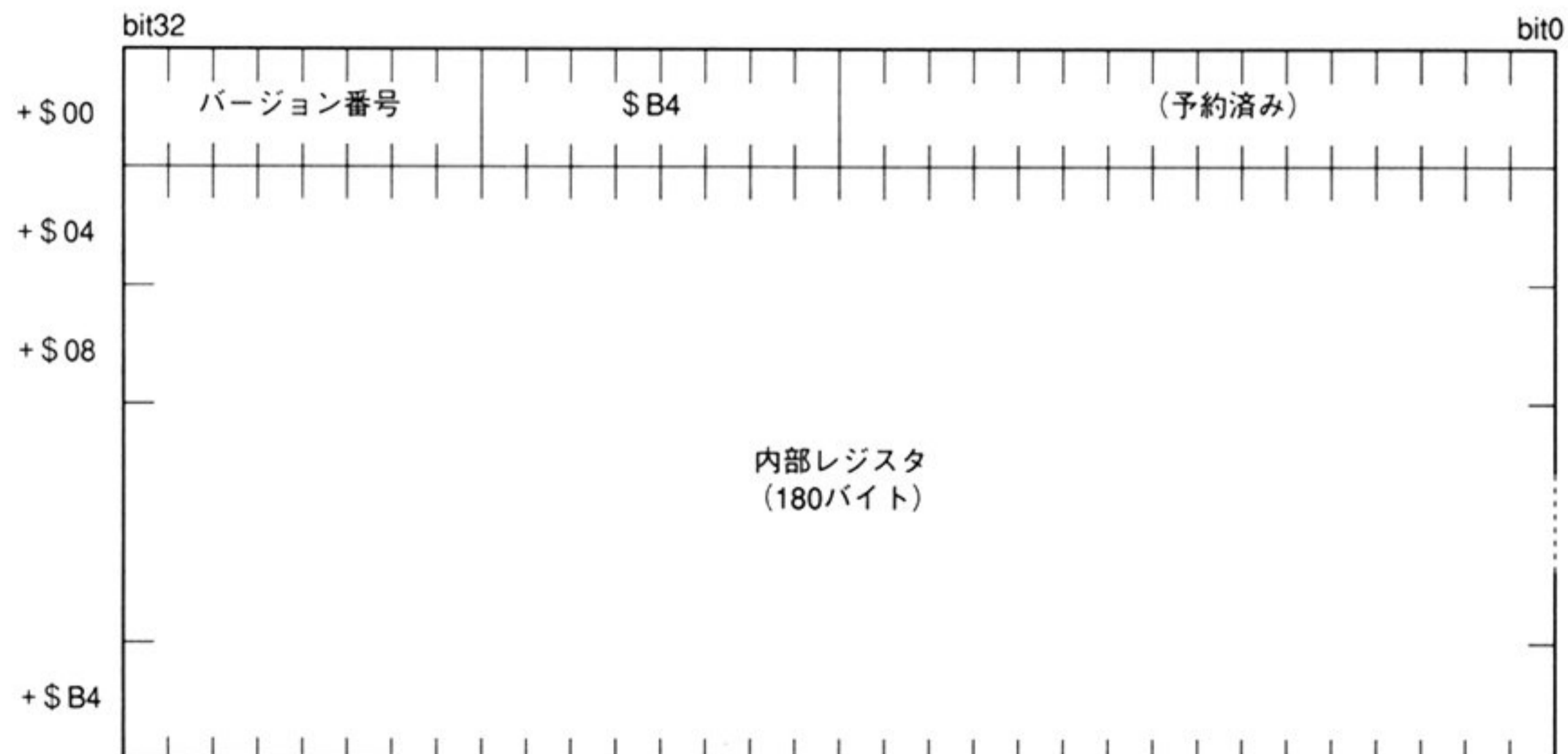
このうち，アイドルステートフレームはMC68881が28バイトであったのに対して，MC68882は60バイト，ビジーステートフレームは184バイトに対して216バイトと，いずれも32バイト増加しています。この増えた分はMC68882で追加されたCUの内部情報です。

アイドルステートフレームとビジーステートフレームは，それぞれステートフレームの2バイト目に(ステートフレームサイズ－4)の値が設定されていますので，MC68881とMC68882の違いとともに見分けることができます。

また，ヌルステートフレームは先頭バイトが$00になっていることで見分けることができま

●図12……ステートフレームのフォーマット

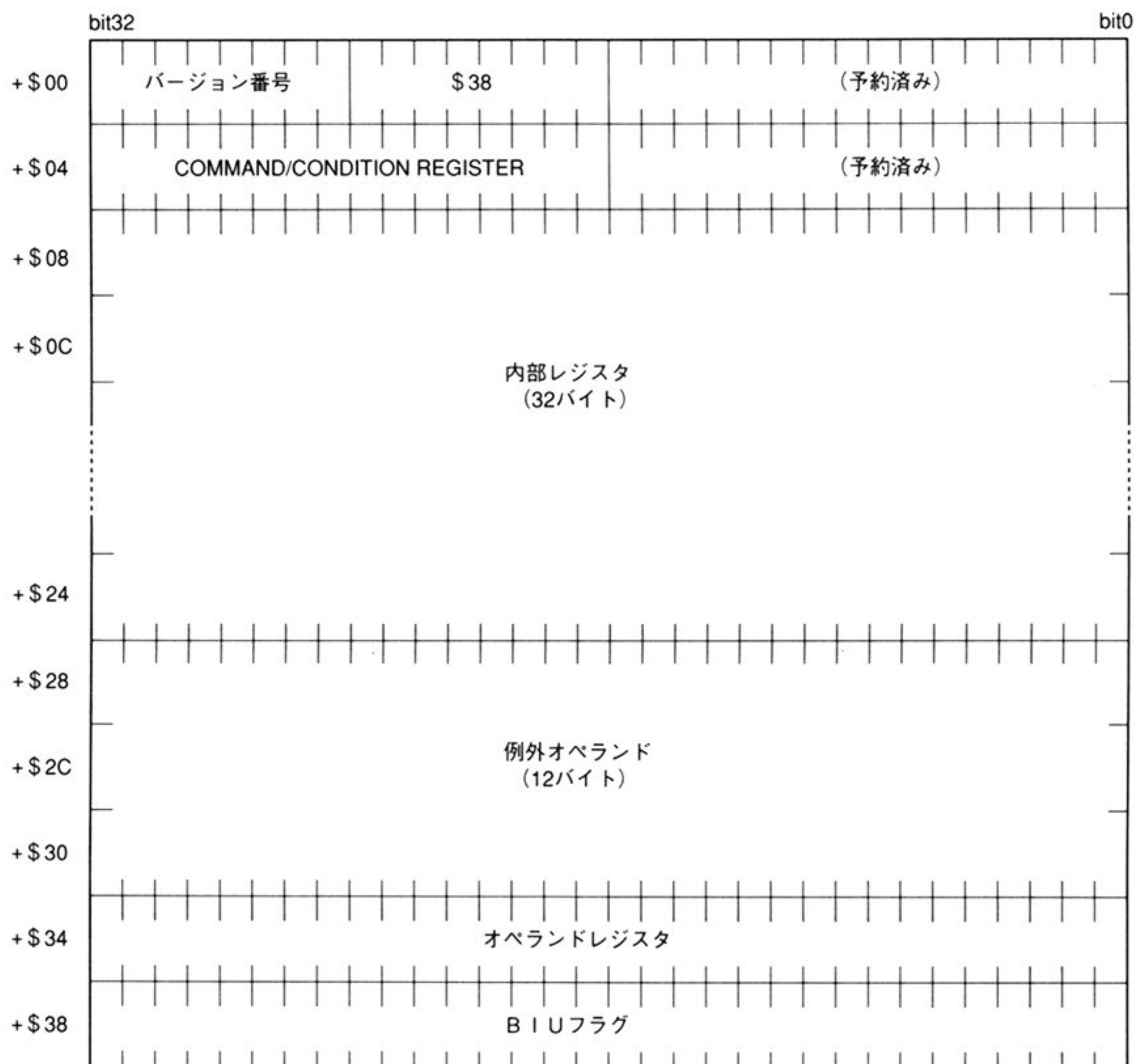
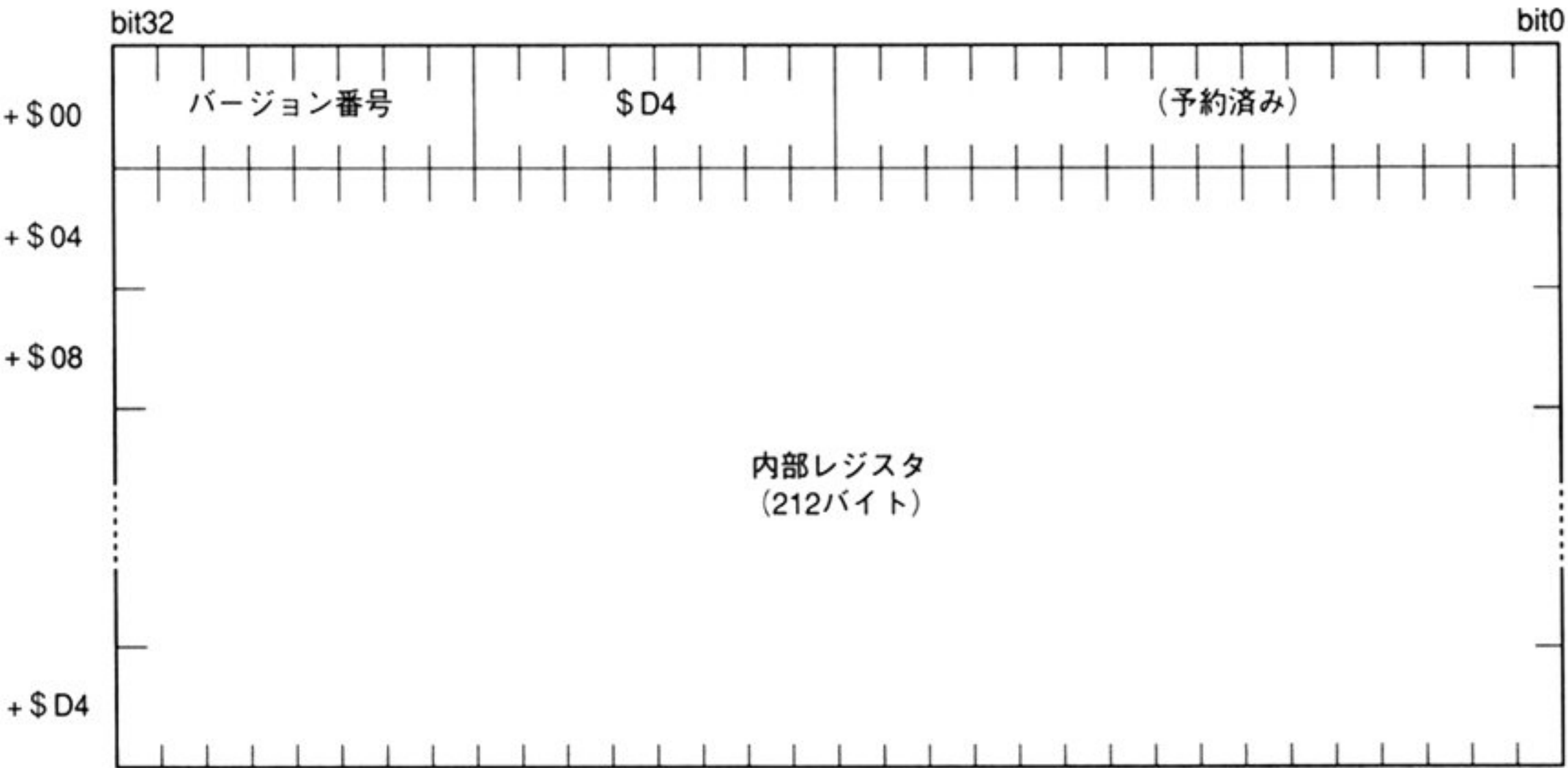
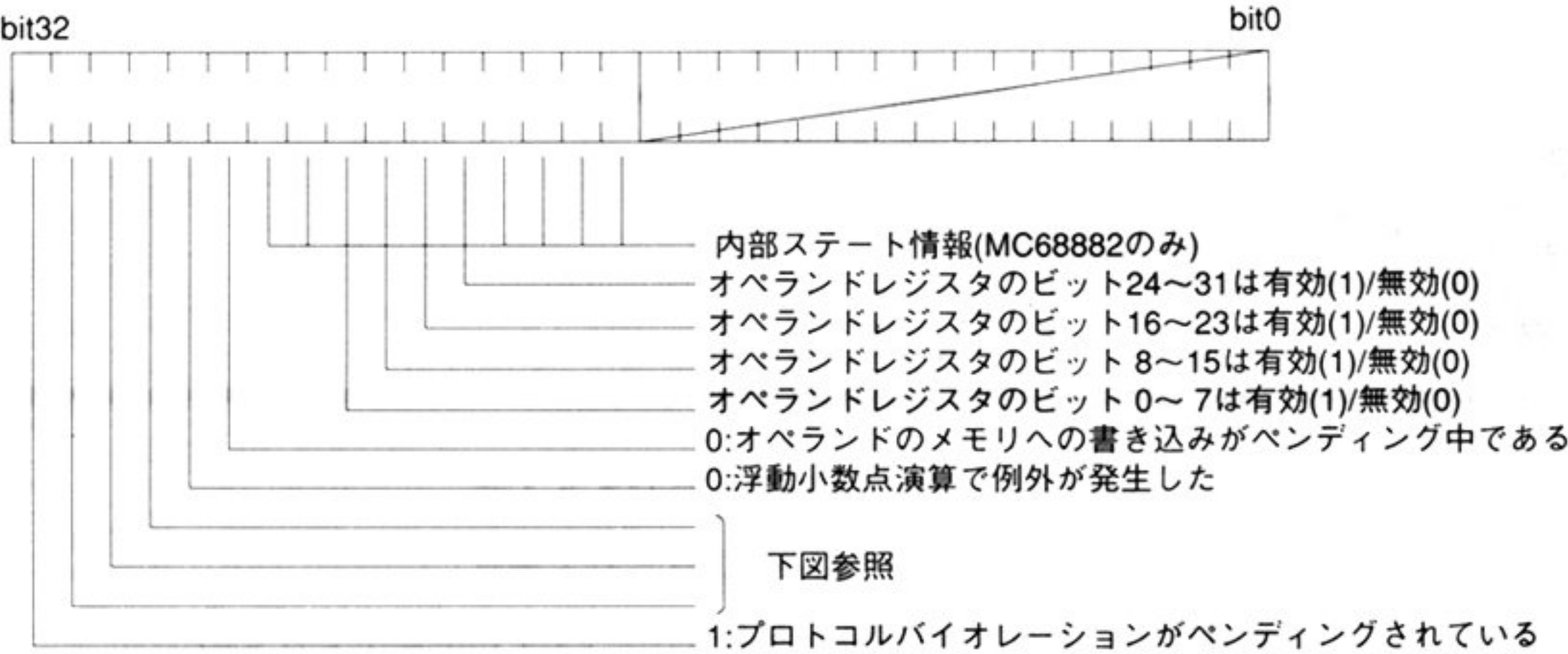
○MC68881の場合

ヌル・ステートフレーム

アイドル・ステートフレーム

ビジー・ステートフレーム

○MC68882の場合

ヌル・ステートフレーム

アイドル・ステートフレーム

オフセット	bit32		bit0
+$00	バージョン番号	$38	(予約済み)
+$04	COMMAND/CONDITION REGISTER		(予約済み)
+$08	内部レジスタ (32バイト)		
+$0C			
⋮			
+$24			
+$28	例外オペランド (12バイト)		
+$2C			
+$30			
+$34	オペランドレジスタ		
+$38	BIUフラグ		

ビジー・ステートフレーム

●図13……BIUフラグのビット配置

bit30	bit29	bit28	内　容
0	0	0	(未定義:モトローラ予約)
0	0	1	条件付き命令がペンディングされている
0	1	0	(未定義:モトローラ予約)
0	1	1	一般的な命令(演算／転送など)がペンディングされている
1	0	0	オペランドCIRへの書き込みがペンディングされている
1	0	1	(未定義:モトローラ予約)
1	1	0	オペランドCIRの読み出しがペンディングされている
1	1	1	ペンディングされている命令やオペランドCIRアクセスはない

す。アイドルステートフレームやビジーステートフレームの先頭バイトは数値演算プロセッサのバージョン番号で$00になることはありません。

アイドルステートフレームの最後にある1ワードのデータは「BIUフラグ」と呼ばれるものです。このフラグのビット配置を87ページの図13に示します。BIUフラグは例外処理などで使用されます。BIUフラグのうち，ビット27以外は書き換えないようにしてください。

3・3 プロトコルバイオレーションの発生条件の違い

詳細は4・1・8のプロトコルバイオレーションの項で説明しますが，MC68881とMC68882ではプロトコルバイオレーションの発生条件が異なることに注意してください。最も問題となりそうな違いは，数値演算プロセッサ側からプログラムカウンタの引き渡しが要求された場合（レスポンスCIRのPCビットが‘1’になった場合），MC68881では要求を無視して次の動作に入ってもかまわなかったのが，MC68882ではプロトコルバイオレーションになり，演算処理が中断されてしまうということです。

X68000シリーズの数値演算プロセッサとしてMC68882を使用する場合には，数値演算プロセッサからのプログラムカウンタの引き渡し要求があったら，必ず命令アドレスCIRへの書き込みを行う必要があります。

4 例外動作

数値演算処理中に発生する例外には，次のようなものがあります。

1.オーバーフローなど値演算プロセッサによって発見される異常

2.CPUによって異常検出されたり，FTRAPcc命令などでCPUが発生するもの

3.トレース例外やアドレスエラーなどCPU自身に起因する異常

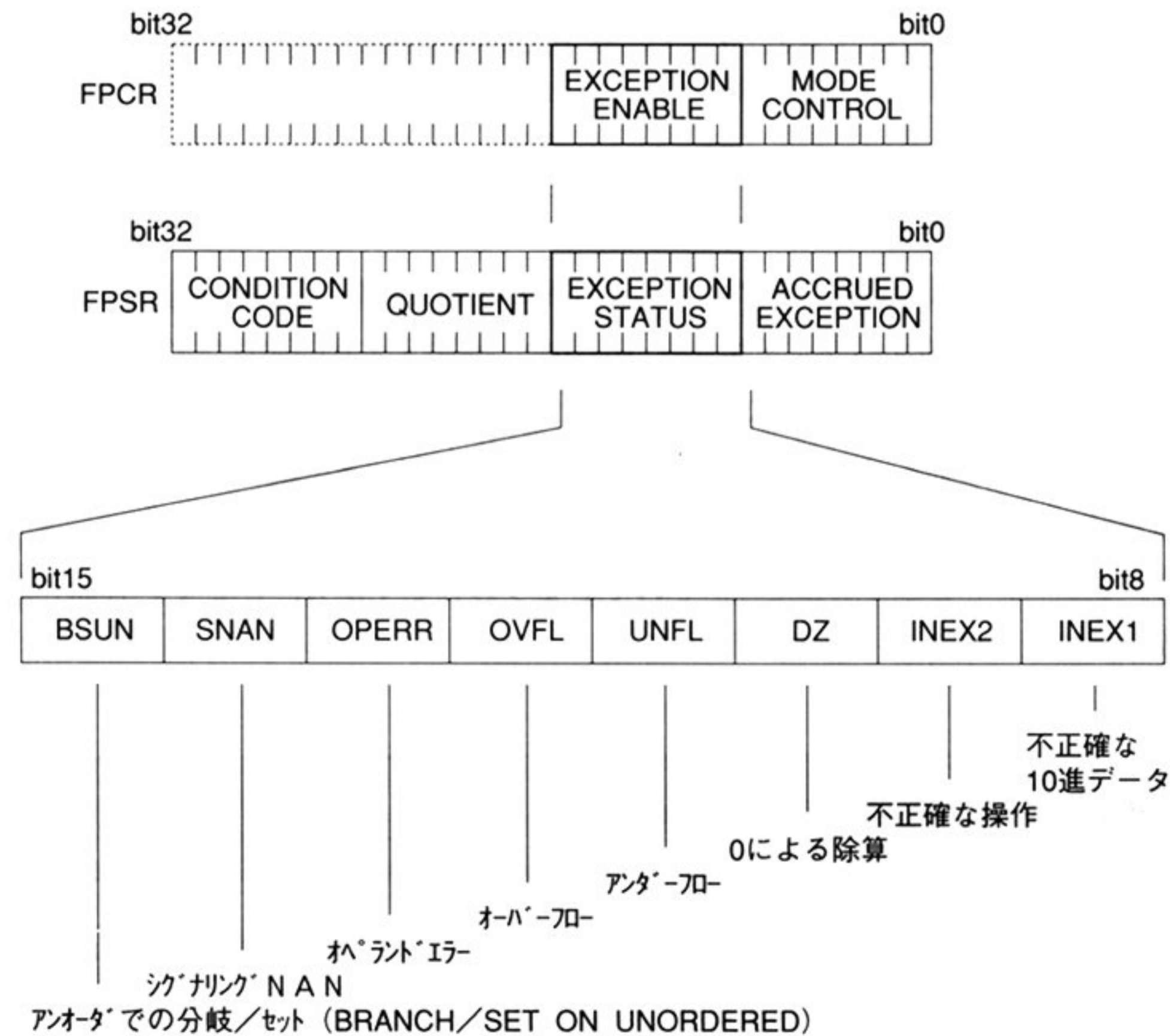
このうち，数値演算プロセッサに固有の例外は1.と2.です。これらの例外が発生したときの例外の種類とベクタ番号の対応関係を図14に示します。10種類の例外のうちTRAPcc例外はCPU, Fラインエミュレータ例外とプロトコルバイオレーションはCPUと数値演算プロセッサの両方，それ以外の8つは数値演算プロセッサが発見するものです。

●図14……数値演算プロセッサと関係する例外とベクタ

ベクタ番号(10進数)	ベクタオフセット値(16進数)	例外種別
7	$01C	FTRAPcc命令
11	$02C	Fラインエミュレータ
13	$034	コプロセッサプロトコルバイオレーション
48	$0C0	BSUN(アンオーダでの分岐/SET命令)
49	$0C4	INEX(不正確な結果)
50	$0C8	浮動小数点ディバイド・バイ・ゼロ
51	$0CC	アンダーフロー
52	$0D0	オペランドエラー
53	$0D4	オーバーフロー
54	$0D8	シグナリングNAN

4・1 数値演算プロセッサが発生する例外

8つの例外のうち，プロトコルバイオレーション以外の7つは数値演算プロセッサの内部レジスタFPCRによって発生の許可／禁止を制御したり，FPSRによって発生した例外を判別す

●図15……FPCR，FPSRの例外処理関連ビット

ることができるようになっています。

これらのレジスタのなかの例外動作に関係するものを図15に示します。同一の命令で複数の例外が発生した場合には，発生したなかで最も上位のビット（BSUNが最上位，INEX 1が最下位）の例外だけが発生し，それ以外のビットは例外を発生しません。このため，数値演算プロセッサの例外が発生したときにはFPSRを見て下位の例外が発生していないかということも確認しておく必要があります。

FPCR，FPSRの全体のビット配置については『Inside X68000』の数値演算プロセッサの章の2・2を参照してください。

4・1・1 BSUN（BRANCH/SET ON UNORDERED）

BSUNは，FBcc，FDBcc，FTRAPcc命令を実行しようとしたとき，前回実行した演算命令の入力データがNAN(Not A Number)であったときに発生します。つまり，演算結果自体が無効となっているので，条件分岐の意味がないわけです。

4・1・2 SNAN（SIGNALLING NOT A NUMBER）

シグナリングNANは，数値演算命令のオペランドがIEEEフォーマットに準拠していないときに発生します。通常，数値演算プロセッサが扱うデータフォーマットはIEEEに準拠しており，数値演算プロセッサの演算の結果でシグナリングNANが発生することはありません。次に説明する数値演算プロセッサのオペランドエラー（0÷0や，無限大×0の演算など）で生成されるNANも，ノン・シグナリングNANだけです。

モトローラのマニュアルでは，この例外はIEEE非準拠の数値データを扱うときに便利であると書いてありますが，たまたま独自フォーマットの数値データがIEEE準拠になってしまった場合はIEEEフォーマットとして演算処理がされてしまうわけですから，どこまで有効であるのかは疑問が残るところでもあります。

4・1・3 OPERR（OPERAND ERROR）

OPERRは，図16に示すような算術的に意味がなかったり，処理できる範囲を越えたオペランドが与えられた場合に発生します。この例外がマスクされているとき，ディスティネーショ

●図16……オペランドエラーが発生する場合

命令	オペランドエラー発生条件
FACOS	(ソース)＝±無限大，(ソース)> +1，(ソース)< -1　のいずれか
FADD	(+無限大)+(−無限大)，(−無限大)+(+無限大)　のいずれかを行った
FASIN	(ソース)＝±無限大，(ソース)> +1，(ソース)< -1　のいずれか
FATANH	(ソース)＝±無限大，(ソース) > +1，(ソース)< -1　のいずれか
FCOS	(ソース)＝±無限大
FDIV	0÷0，(無限大)÷(無限大)　のいずれかを行った
FGETEXP	(ソース)＝±無限大
FGETMAN	(ソース)＝±無限大
FLOG10	(ソース)＝−無限大，(ソース)< 0　のいずれか
FLOG2	(ソース)＝−無限大，(ソース)< 0　のいずれか
FLOGN	(ソース)＝−無限大，(ソース)< 0　のいずれか
FLOGNP1	(ソース)＝−無限大，(ソース)< 0　のいずれか
FMOD	FPnの値が±無限大，またはソースが0で他方がNANでない
FMOVE to (.B/.W/.L)	オーバーフロー，アンダーフロー，ソースがノン・シグナリングNAN，(ソース)= ±無限大のいずれか
FMOVE.P to	指数が999を超えたか，またはKファクタが17を超えたとき
FMUL	片方のオペランドが0で，もう一方が±無限大のとき
FREM	FPnの値が±無限大，またはソースが0で他方がNANでない
FSCALE	ソースが±無限大で，もう一方のオペランドがNANでない
FSGLDIV	0÷0，(無限大)÷(無限大)　のいずれかを行った
FSGLMUL	片方のオペランドが0で，もう一方が無限大のとき
FSIN	(ソース)＝±無限大
FSINCOS	(ソース)＝±無限大
FSQRT	(ソース)< 0，(ソース)＝−無限大　のいずれか
FSUB	ソースとFPnの値がともに+無限大，ないし−無限大の場合
FTAN	(ソース)＝±無限大

ンが浮動小数点レジスタであれば，レジスタにはノン・シグナリングNANがセットされます。

4·1·4 OVFL（OVERFLOW）

演算の途中結果が大きくなりすぎて指定されたフォーマットでは表現しきれなくなった場合に発生します。浮動小数点レジスタ間演算ではFPCRによって指定した丸め精度で表現しきれなくなるとオーバーフローになります。オーバーフローが発生するのは，ディスティネーションがS，D，Xフォーマットのときだけで，整数（B，W，L）やP（パックトBCD）フォーマットの場合にはオペランドエラー扱いになります。

4·1·5 UNFL（UNDERFLOW）

OVFLとは逆に，演算の途中結果が小さすぎて指定されたフォーマットでは表現しきれなくなった場合に発生します。アンダーフローが発生するのは，ディスティネーションがS，D，Xフォーマットのときだけで，P（パックトBCD）のときにはオペランドエラー扱いになります。整数（B，W，L）の場合には0が入るだけで，オペランドエラーにもアンダーフローにもなりません。

4·1·6 DZ（DIVIDE BY ZERO）

DZは，0で除算を行った場合のほか，超越関数を漸化式を使って求めている途中で結果が無限大になってしまったようなときにも発生します。ディバイド・バイ・ゼロが発生する可能性のある演算とオペランドの一覧を図17に示します。0÷0の場合にはディバイド・バイ・ゼロではなくオペランドエラーになるなど，他の例外となることもありますので気をつけてください。

●図17……ディバイド・バイ・ゼロが発生する条件

命令	ディバイド・バイ・ゼロ発生条件
FTANH	(ソース)= ±1
FDIV	(ソース)= 0 で，かつ　FPnがNAN，無限大，0のいずれでもない
FLOG10	(ソース)= 0
FLOG2	(ソース)= 0
FLOGN	(ソース)= 0
FLOGNP	(ソース)= −1
FSGLDIV	(ソース)= 0 で，かつ　FPnがNAN，無限大，0のいずれでもない

4·1·7 INEX2（INEXACT OPERATION），INEX1（INEXACT DECIMAL INPUT）

INEX1は，P（パックトBCD）フォーマットからの変換時に精度落ちが生じて不正確な結果となるとき，INEX2はそれ以外の要因で不正確な結果となるときに発生します。

たとえば，

```
FDIV.P #7E-1 ,FP3
```

のような演算を行う場合，7E－1（0.7）は2進数には正確に変換できないため，INEX1が発生します。また，それに続く割算も不正確な結果になる場合があり，この場合，INEX2が発生します。

INEX2は，演算途中結果の仮数部の有効桁数が多すぎて指定されたフォーマット（Pを除く）では表現しきれなくなった場合にも発生します。

4・1・8 プロトコルバイオレーション（Protocol Violation）

コプロセッサがホストCPUからのアクセスの手順に異常を発見した場合に発生します。MC68881とMC68882ではプロトコルバイオレーションの発生条件が少し異なっています。それぞれの数値演算プロセッサがプロトコルバイオレーションを発生する条件は次のようになっています。

＜MC68881の場合＞

1）コマンドCIRかコンディションCIRに対する書き込みが行われるべきところで，レジスタセレクトCIRやオペランドCIRに対するアクセスが行われた。
2）レジスタセレクトCIRかオペランドCIRの読み出しが行われるべきところで，コマンドCIR，コンディションCIR，オペランドCIRのいずれかに書き込みが行われた。
3）オペランドCIRへの書き込みが行われるべきところで，コマンドCIRやコンディションCIRへの書き込み，またはレジスタセレクトCIRやオペランドCIRの読み出しが行われた。

＜MC68882の場合＞

1）コマンドCIRかコンディションCIRへの書き込みが行われるべきところで，レジスタセレクトCIRやオペランドCIRへのアクセス，または命令アドレスCIRへの書き込みが行われた。
2）レジスタセレクトCIRやオペランドCIRの読み出しが行われるべきところで，コマンドCIR，コンディションCIR，オペランドCIR，命令アドレスCIR，レジスタセレクトCIRのいずれかに書き込みが行われた。
3）オペランドCIRへの書き込みが行われるべきところで，コマンドCIRやコンディションCIRへの書き込みや，レジスタセレクトCIR，オペランドCIR，命令アドレスCIRのいずれかの読み出しが行われた。
4）命令アドレスCIRへの書き込みが行われるべきところで，コマンドCIR，コンディションCIRへの書き込みや，オペランドCIR，レジスタセレクトCIRへのアクセスが行われた。

プロトコルバイオレーションについてはいくつか注意を要する動作がありますので，簡単に説明しておきましょう。

(1)回復不可能なプロトコルバイオレーション

MC68882を使用している場合，回復不可能なプロトコルバイオレーション(Unrecoverable Protocol Violation)となる場合があります。この状態になるのは，FMOVE命令やFMOVEM命令で数値演算プロセッサから外部へのデータ転送を行うとき，実効アドレスを評価する前にオペランドCIRの読み出しを行い，プロトコルバイオレーションとなった場合です。

このとき，MC68882はプロトコルバイオレーションを報告せず，CPUからのアクセスを完全に無視してしまいます。このため，CPUはまったく動作できなくなってしまいます。

(2)意味のないCIRアクセスによるプロトコルバイオレーション

プロトコルバイオレーションは通常，意味のないアクセス(CIRの予約領域へのアクセス，書き込み専用レジスタの読み出し，レジスタセレクトCIR以外の読み出し専用レジスタへの書き込み)では発生しません。唯一の例外はレジスタセレクトCIRで，このレジスタへの書き込みを行った場合にはプロトコルバイオレーションになります。

(3)PC（プログラムカウンタ）の引き渡し

浮動小数点プロセッサは，レスポンスCIRのPCビットを‘1’にして，プログラムカウンタの値の引き渡しを要求してくることがあります。

MC68881はつねにシーケンシャルにしか命令を実行できないので，コプロセッサからのPCの引き渡し要求を無視してもかまいません。しかし，MC68882の場合には命令の平行処理を行えるようになっているため，PCの引き渡しが要求されたときには必ず命令アドレスCIR経由でPCの値を与えないとプロトコルバイオレーションになります。MC68020/MC68030は要求されたときには必ずPCを引き渡すようになっていますので，特に注意する必要はありません。

X68000シリーズにMC68882を取り付けるときはソフトウェアで必ず（値はダミーにすぎませんが）命令アドレスCIRに書き込みを行う必要があります。

(4)プロトコルバイオレーションの見分け方

プロトコルバイオレーションは数値演算プロセッサとCPUのどちらでも検出されます。プロトコルバイオレーションが発生したとき，それが数値演算プロセッサとCPUのどちらで発見されたものであるのかを判別するには，例外処理ルーチンのなかでレスポンスCIRを読めばわかります。もし数値演算プロセッサが発見していれば，発生した例外がCPUに受け取られた時点で数値演算プロセッサは処理を中断し，アイドル状態に復帰しています。したがって，レスポ

ンスCIRを読み出すとヌル（CAビット＝‘0’）になっているはずです。

もしCPUが発見したのであれば，CPUがプロトコルバイオレーションであると判断した時点のレスポンスCIRの値が読み出せるはずです。ただ，数値演算プロセッサは異常なプリミティブを返すことはありませんので，CPUがプロトコルバイオレーションを発生した場合は，ハードウェアの異常が最も疑わしいと思われます。

レスポンスCIRを読み出すには次のような方法を使えばよいでしょう。

```
MOVE.B    #7,D0
MOVEC     D0,SFC
MOVES.W   $00022000,D0
```

最初の２行で，最後のMOVES.W命令でアクセスする空間をCPU空間に設定しておき，最後の命令でレスポンスCIRを読み出します。３行目で使っているアドレスは数値演算プロセッサのデフォルトのアドレスです。ビット19～16（‘0010’）はコプロセッサとのコミュニケーションであることを示す固定データ，ビット15～13（‘001’）はコプロセッサIDです。数値演算プロセッサのコプロセッサIDは１ですので‘001’となります。ビット１～４はCIRの選択に使用されます。CIRの配置順は『Inside X68000』の数値演算プロセッサのページを参照してください。レスポンスCIRをアクセスする場合，ビット１～４は‘0000’となります。

4・2 メインプロセッサが検出する例外

メインプロセッサが検出する例外としては，FTRAPccやトレース例外のように意図的に発生させるものと，イリーガル命令のようにCPU自身がなんらかの異常を発見した結果として発生するものがあります。

4・2・1 コプロセッサコンディション命令

FTRAPccのように，数値演算プロセッサの条件判定結果の真偽によって例外処理に入るか否かを決める命令で行われる例外です。TRAPVやTRAPccのベクタが使用されます。

4·2·2 イリーガル命令

コプロセッサ命令のコードには，CPUに対するオペレーションワードとコプロセッサに対するコマンドワードが含まれています。CPUがコプロセッサ用のオペレーションワードの異常や使用できないアドレッシングモードを指定しているなどの異常を発見すると，例外が発生します。このときのベクタには，Fラインエミュレータ例外のベクタが使用されます。

4·2·3 プロトコルバイオレーション

CPUが数値演算プロセッサから読み出したレスポンスを異常と判断した場合，プロトコルバイオレーション例外が発生します。数値演算プロセッサ自身は正常なレスポンスしか返さないので，この異常が発生したときにはハードウェア的な異常が起こったことが疑われます。

4·2·4 フォーマットエラー

FRESTORE命令を実行するとき，数値演算プロセッサは与えられたステートフレームが正当なものであるか否かのチェックを行っています。このとき，異常が見つかるとフォーマットエラー例外となります。

また，数値演算プロセッサがFSAVEやFRESTORE命令を実行しているときにFSAVE命令が実行されたときもフォーマットエラー例外になります。

4·3 例外処理

MC68882が例外を発生した場合の基本的な処理方法は次のようになります。

1. FSAVE命令でステートフレームを取り出す
2. BSET命令などでステートフレーム中のBIUフラグのEXCPENDビットを‘1’にする
3. FRESTORE命令でステートフレームをMC68882に戻す

この処理は，FPSRレジスタの例外ステータスバイトに反映される要因（アンオーダでの分岐／セット，不正確な結果，0除算，アンダーフロー，オーバーフロー，オペランドエラー，シグナリングNAN）の例外処理の先頭で必ず行う必要があります。これ以外の例外（Fライン

エミュレータ例外やFTRAPcc命令など)の場合には，2.のEXCPENDビットの操作は行わないようにしてください。

例外処理のなかで浮動小数点命令を実行したり，浮動小数点演算を行う別のタスクへの切り替えなどを行う場合には必ずFSAVE命令でステートフレームを退避して，元のプログラムに戻るときにFRESTOREする必要があります。

最も単純な例外処理のプログラムは次のようになります。

```
        FSAVE    -(SP)           * ステートフレームの退避
        MOVE.B   (SP),D0
        BEQ      NOTHINGTODO     * ヌルステートフレームなら何もしない
        CLR.L    D0
        MOVE.B   1(SP),D0        * BIUフラグの位置を得る
        BSET     #3,(SP,D0)      * EXCPENDビットのセット
          .
          .                      * そのほか，必要な処理があればここに入れる
          .
NOTHINGTODO:
        FRESTORE (SP)+           * ステートフレームの回復
        RTE
```

5 サンプルプログラム

MC68882の最大の特徴といえる内部の平行動作の効果がどの程度あるものかを確認するプログラムと，サンプルで使用した数値演算プロセッサ用の命令を68000用アセンブラでアセンブルできるようにするための簡単なフィルタプログラムを作ってみました。

●リスト　簡易版数値演算プロセッサ命令トランスレータ

```
/*
 *  簡易版数値演算プロセッサ用命令トランスレータ
 *
 *  コプロセッサ命令に対応していないアセンブラでサンプルプログラム
 *  をアセンブルするための簡易トランスレータ
 *
 * cpt < src.s > dst.s
 *
```

```
*<制約事項>
*  命令はfmove, fadd, fmulのみ
*  ディスティネーションオペランドはFPn, (An), (An)+, -(An)のみ
*  ソースオペランドは、FPn, (An), (An)+, -(An), イミディエイトデータのみ
*    イミディエイトデータは浮動小数点表記のみ
*  データフォーマットはS, Dのみ
*
*/

#define OPNUM   3   /* 演算命令の種類             */
#define SZNUM   8   /* サイズフィールドの種類        */
#define SRCNUM  33  /* ソースオペランドの種類        */
#define DSTNUM  32  /* ディスティネーションオペランドの種類 */
#define SRC_IMM (SRCNUM-1)
#define EOS '¥0'

union   DATAS {
    float        fval;
    unsigned long   ifval;
} datas;

union   DATAD {
    double       dval;
    unsigned long   idval[2];
} datad;


struct  TOKEN {
    unsigned int    token_num;
    unsigned char   *token_val[40];
};

struct TOKEN    op = {
    OPNUM,
    "fmove", "fadd", "fmul"
};

struct TOKEN     sz = {
    SZNUM,
    ".l", ".s", ".x", ".pk", ".w", ".d", ".b", ".pd"
};

struct  TOKEN   src = {
    SRCNUM,
    "fp0", "fp1", "fp2", "fp3", "fp4", "fp5", "fp6", "fp7",
    "(a0)", "(a1)", "(a2)", "(a3)", "(a4)", "(a5)", "(a6)", "(sp)",
    "(a0)+", "(a1)+", "(a2)+", "(a3)+", "(a4)+", "(a5)+", "(a6)+", "(sp)+",
    "-(a0)", "-(a1)", "-(a2)", "-(a3)", "-(a4)", "-(a5)", "-(a6)", "-(sp)",
    "#"
};

struct  TOKEN   dst = {
    DSTNUM,
    ",fp0", ",fp1", ",fp2", ",fp3", ",fp4", ",fp5", ",fp6", ",fp7",
```

```
    ".(a0)",".(a1)",".(a2)",".(a3)",".(a4)",".(a5)",".(a6)",".(sp)",
    ".(a0)+",".(a1)+",".(a2)+",".(a3)+",".(a4)+",".(a5)+",".(a6)+",".(sp)+",
    ".-(a0)",".-(a1)",".-(a2)",".-(a3)",".-(a4)",".-(a5)",".-(a6)",".-(sp)"
 };

unsigned int    op_val[] = {          /* 演算命令コード        */
    0x00,0x22,0x23
};

unsigned char   *sp;                  /* 入力文字列へのポインタ   */
unsigned char   buf[256];             /* 入力バッファ          */

void main();
int fcode();
void gen_code();
unsigned int get_token();
unsigned char *smatch();
double  get_value();
int gen_fmove();
int gen_etc();

void main()
{
    while(1) {
        if (!gets(buf))
            exit(0);
        sp = buf;
        if (fcode())
            printf("%s¥n",buf);
        else    printf("* %s¥n",buf);
    }
}

int fcode()
{
    unsigned int    opn,szn,srcn,dstn;
    double  dat;
    opn = get_token(sp,&op);
    if (opn == 0xffff)
        return(1);
    szn = get_token(sp,&sz);
    if (szn == 0xffff)
        return(1);
    srcn = get_token(sp,&src);
    if (srcn == 0xffff)
        return(1);
    if (srcn == SRC_IMM) {        /*  イミディエイトなら数値を読む */
        dat = get_value(sp);
        datas.fval = dat;
    }
    dstn = get_token(sp,&dst);
    if (dstn == 0xffff)
        return(1);
    gen_code(opn,szn,srcn,dstn,dat);      /* dc.w/dc.lなどに展開 */
```

```
    return(0);
}

unsigned int get_token(cmd, tbl)
    unsigned char   *cmd;
    struct  TOKEN   *tbl;
{
    unsigned int    i;
    unsigned char   *p;
    for (i=0; i<tbl->token_num; i++) {
        if (p = smatch(cmd, tbl->token_val[i])) {
            sp = p;
            return(i);
        }
    }
    return(0xffff);
}

unsigned char *smatch(s, d)
    unsigned char   *s, *d;
{
    while((*s == ' ') || (*s == '¥t'))
        *s++;
    while(1) {
        if (*d == EOS)
            break;
        if (*d++ != *s++)
            return((unsigned char *)0);
    }
    if (*s == '+')      /* (a0)+対応のため  */
        return((unsigned char *)0);
    return(s);
}

double  get_value(p)
    unsigned char   *p;
{
    unsigned int    iv;
    double  d, s;
    d = 0.0;
    while(1) {
        if (*p == '.')
            break;
        if ((*p < '0')||(*p > '9')) {
            sp = p;
            return(d);
        }
        d = d*10.0+(*p-'0');
        p++;
        }
        p++;
        s = 0.1;
        while(1) {
            if ((*p < '0')||(*p > '9')) {
```

```
                sp = p;
                return(d);
            }
            d += s*(double)(*p-'0');
            s /= 10.0;
            p++;
        }
}

void gen_code(op,sz,src,dst,dat)
    unsigned int    op,sz,src,dst;
    double      dat;
{
    switch(op) {
        case    0: gen_fmove(op,sz,src,dst,dat);
                break;
        case    1:
        case    2: gen_etc(op,sz,src,dst,dat);
        default:    break;
    }
}


int gen_fmove(op,sz,src,dst,dat)
    unsigned int    op,sz,src,dst;
    double      dat;
{
    unsigned int    opw,cmw;
    if (dst >= 8) {     /* fmove fpn,(An)   */
        if (src >= 8)   /* ｿｰｽｵﾍﾟﾗﾝﾄﾞはfpnのみ  */
            return(-1);
        opw = 0xf200+(dst+0x8);
        cmw = 0x6000+(src << 7)+(sz << 10);
        printf("    dc.w    $%04x,$%04x¥n",opw,cmw);
        return(0);
    }
    else {
        if (src < 8) {      /* fmove fpn,fpm    */
            opw = 0xf200;
            cmw = (src << 10) + (dst << 7);
            printf("    dc.w    $%04x,$%04x¥n",opw,cmw);
        }
        else if (src < 32) {    /* fmove (An),fpnなど   */
            opw = 0xf200 + (src+8);
            cmw = 0x4000 + (dst << 7)+(sz << 10);
            printf("    dc.w    $%04x,$%04x¥n",opw,cmw);
        }
        else {          /* fmove #xxx,fpn   */
            opw = 0xf23c;
            cmw = 0x4000 + (dst << 7)+(sz << 10);
            printf("    dc.w    $%04x,$%04x¥n",opw,cmw);
            if (sz == 1) {
                datas.fval = dat;
                printf("    dc.l    $%08lx¥n",datas.ifval);
```

```
            }
            else {
                datad.dval =dat;
                printf("      dc.l    $%08lx,$%08lx¥n",datad.idval[0],datad.idval
[1]);
            }
        }
    }
    return(0);
}

int gen_etc(op,sz,src,dst,dat)
    unsigned int    op,sz,src,dst;
    double      dat;
{
    unsigned int    opw,cmw;
    if (src < 8) {      /* fxxxx fpn,fpm    */
        opw = 0xf200;
        cmw = (src << 10) + (dst << 7)+op_val[op];
        printf("      dc.w    $%04x,$%04x¥n",opw,cmw);
    }
    else if (src < 32) {    /* fxxxx (An),fpnなど   */
        opw = 0xf210 + (src -8);
        cmw = 0x4000 + (dst << 7)+(sz << 10) + op_val[op];
        printf("      dc.w    $%04x,$%04x¥n",opw,cmw);
    }
    else {          /* fxxxx #xxx,fpn   */
        opw = 0xf23c;
        cmw = 0x4000 + (dst << 7)+(sz << 10) + op_val[op];
        datas.fval = dat;
        printf("      dc.w    $%04x,$%04x¥n",opw,cmw);
        printf("      dc.l    $%08lx¥n",datas.ifval);
    }
}
```

●リスト　数値演算プロセッサ用の命令を，68000用アセンブラでアセンブルするためのフィルタ

```
/*
 * MC68882の平行動作機能の実験
 *      x(i) = y(i)*C + x(i);
 * コメント化してあるものと差し替えるとレジスタの衝突のため、
 * 平行動作ができなくなります。
 *
 * インラインアセンブラでレジスタを壊しているため、gccでコン
 * パイルするときは-finline-functionsオプションは付けないよう
 * にしてください。
 *
 */
double  datx0[1000];
double  daty0[1000];
double  datx1[1000];
double  daty1[1000];
```

```
long    t;

void    main();
long    time0();
long    time1();
void    set_val();

void main()
{
    set_val();
    SUPER(0);
    t = time0();
    asm ("  move.l #$101,d0");  /* キャッシュON */
    asm ("  dc.w $4e7b");       /* (ユーザモードでキャッシュが */
    asm ("  dc.w $0002");       /* OFFになっているときにSUPERで */
    asm ("  move.l  #1000,d1"); /* スーパバイザモードになっても */
    asm ("LOP:");            /* キャッシュがOFFのままになって*/
    asm ("  move.l  #999,d0");  /* いたため)        */
    asm ("  lea.l   _datx0,a0");
    asm ("  lea.l   _daty0,a1");
    asm ("  lea.l   _datx1,a2");
    asm ("  lea.l   _daty1,a3");
    asm ("  fmove.d #3.14159265,fp0;");
    asm ("  fmove.x fp0,fp1");
    asm ("  fmul.d  (a1)+,fp1");
    asm ("  bra LOOP_START");
    asm ("LOOP_TOP:");
    asm ("  fmove.x fp0,fp1");  /* fmove.d fp1,(a2)+    */
    asm ("  fmul.d  (a1),fp1"); /* fmove.x fp0,fp1  */
    asm ("  fmove.d fp2,(a2)+");    /* fmul.d  (a1),fp1 */
    asm ("LOOP_START:");
    asm ("  fadd.d  (a0),fp1");
    asm ("  fmove.x fp0,fp2");  /* fmove.d  fp1,(a0)+   */
    asm ("  fmul.d  (a3)+,fp2");    /* fmove.x  fp0,fp1 */
    asm ("  fmove.d fp1,(a0)+");    /* fmul.d   (a3)+,fp1   */
    asm ("  fadd.d  (a2),fp2");
    asm ("  dbra    d0,LOOP_TOP");
    asm ("  fmove.d fp2,(a2)");
    asm ("  dbra    d1,LOP");
    asm("move.l #$0,d0");       /* キャッシュOFF    */
    asm("dc.w $4e7b");      /* (ONのまま終了するとディスク */
    asm("dc.w $0002");      /* アクセスなどで異常がおきる   */
    printf("%ld秒経過¥n",time1()-t);/* ことがある)        */
}

long time0()
{
    return(time((long)0));
}

long time1()
{
    return(time((long)0));
}
```

```
void set_val()
{
    unsigned int    i;
    for (i=0; i<1000; i++) {
        datx0[i] = (double)i / 1000.0;
        daty0[i] = (double)i / 1000.0;
        datx1[i] = (double)i / 1000.0;
        daty1[i] = (double)i / 1000.0;
    }
}
```

まず，フィルタの使い方を簡単に説明しておきましょう。gccでコンパイルするときに-Sオプションをつけて，アセンブラのソースを出力させます。このソースを

```
cpt < cpcmp.s > t.s
```

というようにしてcptを通すと，fmoveなどの命令がDC.Wなどで展開されたt.sというファイルを得られます。後はCのプログラムをコンパイルするときと同じように

```
cc t.s
```

とすれば，t.xという実行ファイルが得られます。

サンプルで行っている演算は積和演算で，MC68882のマニュアルのあちこちで平行動作のサンプルとして取り上げられているので，そのまま使用してみました。コメントアウトしてあるものは，命令順と演算に使用するレジスタをfp1のみにすることでシーケンシャルにしか処理が進まないようにしてみたものです。

実行してみると，平行動作をさせたときが12秒程度，シーケンシャル動作のときが16～17秒くらいとなりました。平行動作を意識してプログラムしただけで1.3～1.4倍程度高速化されたわけです。

システムポート

システムポートは，元祖X68000からX68000XVI，X68030と世代が変わるにつれてレジスタや機能の追加が行われています。ここでは，これらで追加されたシステムポートの内容を中心に説明します。

1 システムポートの整理

システムポートについては『Inside X68000』でも説明しましたが，少々説明が不足していたところもありましたし，X68000XVIやX68030で追加されたポートもありますので，ここで再度全レジスタについて説明しておきます。X68030のシステムポートの一覧を図１に示します。システムポートが増設されたのにともない，レジスタ番号が新しく振り直されていますので，注意してください。

システムポート#1

コンピュータ画面のコントラストの調整を行います。下位４ビットで明るさの度合いが決まり，$Fが最も明るく，$0が最も暗くなります。Human68kは通常$Eで使用しており，電源OFFのときには，このポートを使って画面を徐々に暗くし，最後に電源を落とすようにしています。

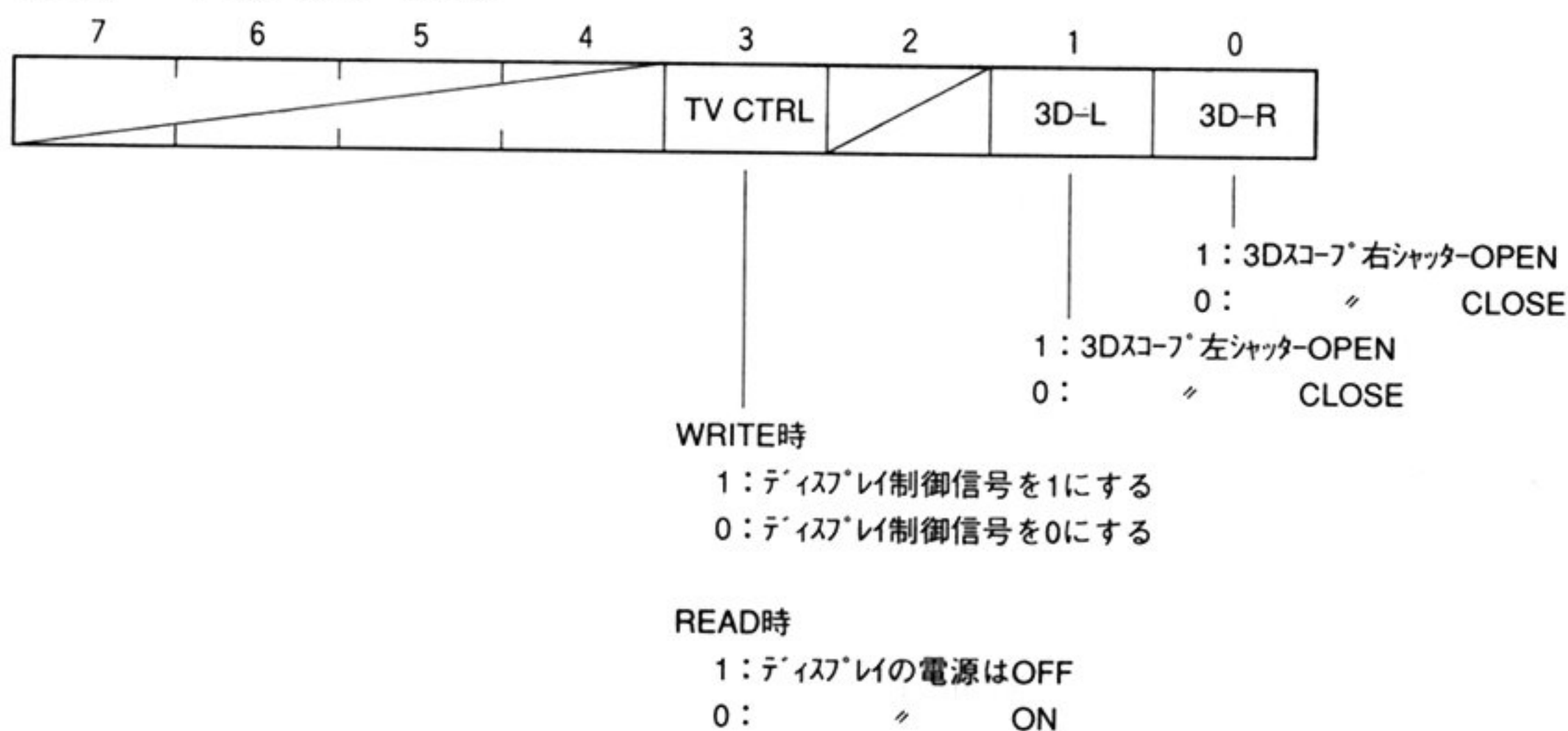
●図1……システムポートのアドレス

レジスタ#	アドレス	bit 7	6	5	4	3	2	1	bit 0	備考
1	$E8E001					CONTRAST				コンピュータ画面コントラスト
2	$E8E003					TV CTRL		3D-L	3D-R	ディスプレイ／3Dスコープ制御
3	$E8E005				カラーイメージユニット制御					
4	$E8E007					KEY CTRL	NMI RESET	HRL		キーボード／NMI／ドットクロック制御
5 *1	$E8E009	ROM Wait Control				DRAM Wait Control				ROM/DRAMウェイト数制御
6 *2	$E8E00B	MPU Type				MPU Clock				MPUの種別，動作クロック読み出し
7	$E8E00D	SRAM Write Enable Control								SRAM 書き込み制御
8	$E8E00F					Power Off Control				本体電源OFF制御

＊1：X68030から追加されたポート
＊2：X68000XVIから追加されたポート
※追加されたポートを，追加される前の機種でアクセスすると$FFが読み出される。

システムポート#2

ビット配置を図2に示します。下位2ビットはオプションの3Dスコープの制御に用いられるものです。ビット0が右目，ビット1が左目のシャッターに対応しており，それぞれ'1'になっているとシャッターがOPENし，画面が見えるようになります。

●図2……システムポート＃2

ビット３は，書き込み時はディスプレイ制御信号，リード時はディスプレイの電源ON/OFFステータスとして動作します。このビットの詳細については『Inside X68000』のキーボードの章を参照してください。

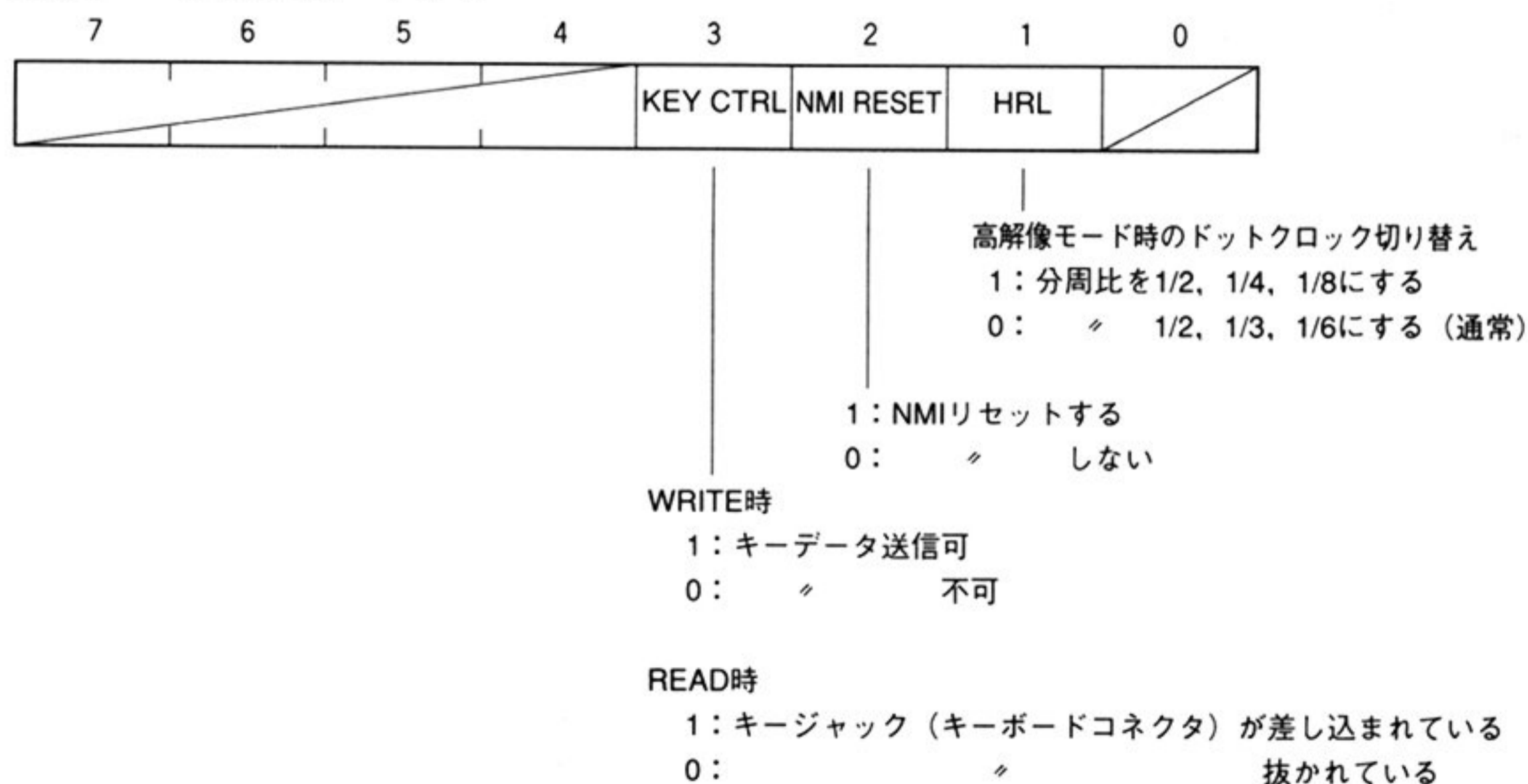
システムポート#3

システムポート#3は，オプションのカラーイメージユニットの制御に使用されるものです。ここに書き込んだ値はそのままIMAGE　IN端子の17～21番ピン（17がビット４，21がビット０に対応）に出力されます。

システムポート#4

ビット配置は図３のようになっています。ビット１はドットクロックの切り替えに使われます。通常は‘０’で使われていますが，‘１’にすると基本周波数の１／４や１／８のクロックが得られます。これを使えば，たとえば，384×256ドットといった特殊な画面モードで動作させることができるようになります。このビットの詳細については本書のCRTCの章を参照してください。

ビット２はNMIが発生したとき，NMIの処理が終了した時点で‘１’を書き込むビットです。このビットに‘１’を書き込まないと，NMI処理から復帰したとたん，再度NMIが入ってしまいます。

●図３……システムポート＃４

ビット3は，キーボードの制御やキーコネクタが差し込まれているか否かのチェックを行うためのビットです。これらのビットの詳細は『Inside X68000』のキーボードの章を参照してください。

システムポート#5

このポートは，X68030から増設されたポートです。ROMやDRAM(メインメモリ)へのアクセス時のウェイト数を制御します。ノーウェイトでは速すぎて困るようなアプリケーション(ゲームソフトなど)があった場合，このポートを使うことでCPUの処理速度を下げることができます。

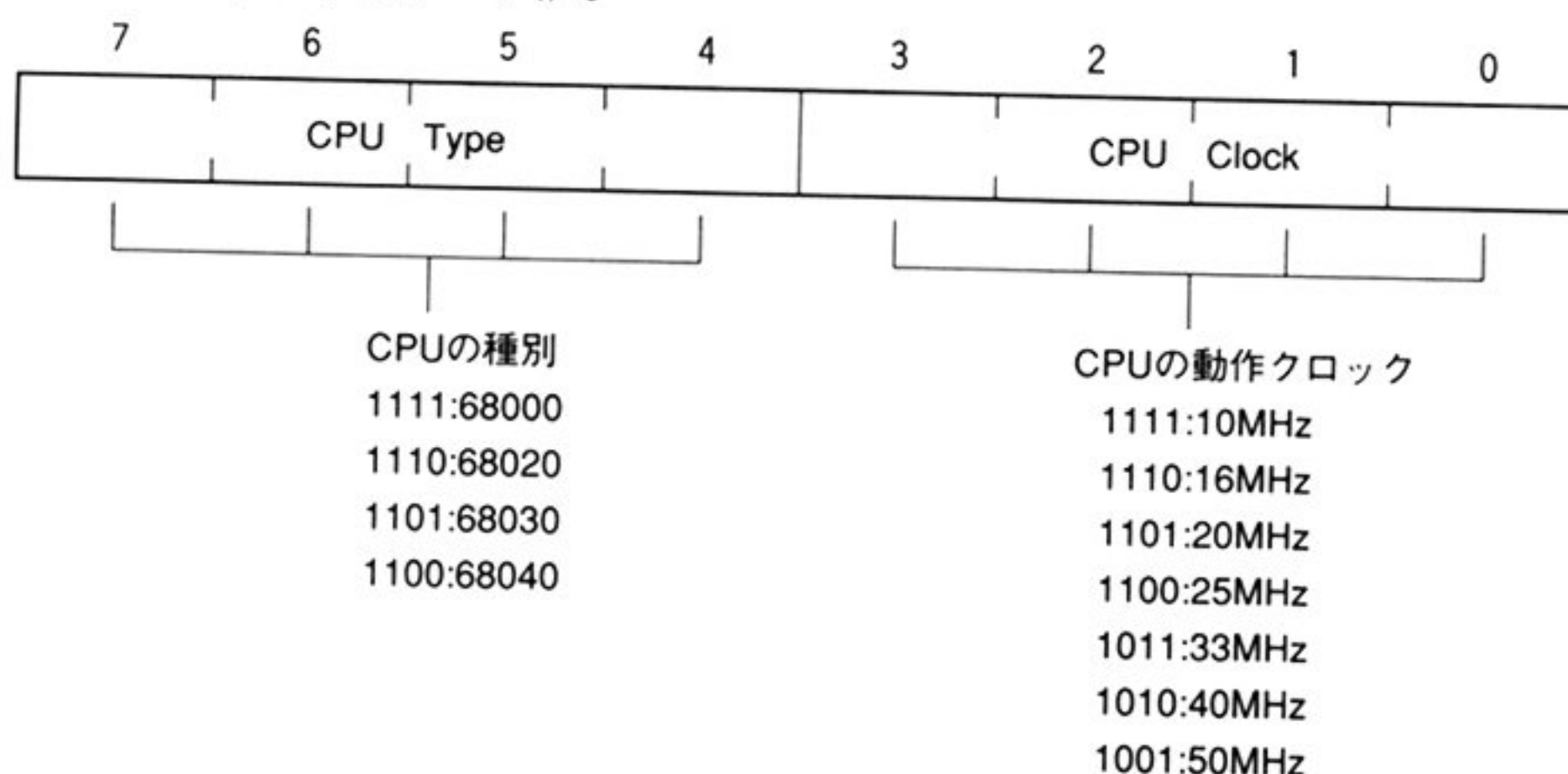
上位4ビットはROMアクセス時のウェイト数，下位4ビットはDRAMへのアクセス時のウェイト数を指定します。いずれも0のときが最高速（ノーウェイト)，$Fが最も低速になります。このポートは書き込み専用のため，読み出すとつねに$FFになっています。

X68030のBIOS-ROMでは，通常立ち上げたときは$00をセットしてノーウェイト動作にしますが，［XF3］キーを押して立ち上げたときにはこのポートに$0Aを，［XF4］キーを押して立ち上げたときは$04をセットして速度をX68000，X68000XVI相当にしています。

このポートの操作によるアクセスタイミングの変化の詳細については，X68030の動作タイミング実測結果の章を参照してください。

システムポート#6

X68000XVIから追加されたポートです。ビット配置を図4に示します。使用されているCPUの種類と動作クロックが読み出されます。上位4ビットがCPUの種別，下位4ビットが動

●図4……システムポート＃6

作周波数を示します。X68000XVI以前の機種では，このポートはありませんでした。読み出すと$FFが読めるため，これでCPUが68000，クロックが10MHzを示すビットパターンとして定義されています。

X68030では，［XF3］や［XF4］キーを押しながら立ち上げると16MHzや10MHz互換になると取扱説明書に記述がありますが，このポートからはつねに$DC(CPU：68030，クロック：25MHz）が読み出されます。

システムポート#7

このポートは，$ED0000～$ED3FFFに配置されている16KバイトのSRAMへの書き込みの許可／禁止を制御するものです。SRAMにはメモリ容量などのシステム情報や起動デバイス，キーボードの文字選択などの情報が保持されている（電源を切っても消えないようにバックアップが行われている）ため，プログラムミスなどがあっても容易に書き換わらないようにしておく必要があります。このために設けられたのがシステムポート#5です。

このポートに$31を書き込むとSRAMへの書き込みが許可になり，それ以外のデータを書き込むと書き込み禁止になります。SRAMの読み出しは常時行えます。

システムポート#8

本体の電源OFFを行うものです。本体正面の電源スイッチがOFFになっているときに，このポートに$00，$0F，$0Fと順に書き込みを行うことで，本体の電源をOFFすることができます。不用意に電源が落ちるのを防ぐため，この順序で書き込まなければ働かないようになっています。アプリケーションソフトでこのポートを使用して電源をOFFにする場合には２点ほど気をつけなければならない点があります。

1) OSの電源OFF処理を行わせない

　電源スイッチOFFの割り込みを禁止，あるいはベクタをフックするなどして，OS側で行っている電源OFF処理を殺しておく必要があります。電源OFFの割り込みは，MFPのGPIP2に入ってきます。

2) リアルタイムクロックのアラーム出力を止める

　Human68kでは，通常動作時にリアルタイムクロックのアラーム出力に1Hzのクロックを出力するように設定しています（RTCのRESETコントローラレジスタに$4を設定しています)。アラーム出力は，タイマ起動動作時の電源ON用としても使われているため，そのままにしておくと，システムポート#8で電源OFFを行っても500msもたたないうちにアラーム出力がONになり，電源が再度ONになってしまいます。アラーム出力を止めるには，まず，リアルタイムクロックのRESETコントローラレジスタのビット２，３を'11'に設定し

ます。タイマー起動モードにしない場合にはさらにアラームイネーブルビット（MODEレジスタのビット2）を‘0’に設定してアラーム動作自体を禁止しておくのが安全でしょう。

システムポート#8を使って電源をOFFにするときには，これらの設定を忘れずに行うようにしてください。

拡張スロット

X68030の拡張スロットは，X68000用のオプションボードができるだけそのまま利用できるように考慮されていますが，やはり若干の違いは生じています。ここではX68030の拡張スロットの動作タイミングとX68000との違いについて説明します。

1 概要

X68030に使われたCPU，MC68EC030のバスアクセスタイミングは，X68000シリーズのCPU（MC68HC000相当品）とはまったく互換性がありません。このため，X68030の内部の動作タイミングはX68000シリーズとは異なるものになっていますが，拡張スロットについては信号の種類，タイミングとも従来のX68000シリーズとほぼ同等のものを作り出して出力するようにしています。これにより，X68030でもX68000用に作られた周辺ボードの多くが使用可能となっています。

ただし，これらの信号はあくまでもX68000シリーズに似せて作っているもので，X68000とまったく同じというわけではありません。また，X68030シリーズではX68000シリーズの周辺ボードではほとんど使われることのなかった信号の廃止やほかの信号への割り振り，信号の動作の変更といった改訂が行われています。

ここでは，まずX68030で変更や廃止の行われた信号について説明した後，拡張スロットの主要な動作タイミングについて説明することにします。

なお，変更されていない信号の意味などについては『Outside X68000』を参照してください。

2 変更／廃止された信号

X68030で変更／廃止された信号の一覧を図１に示します。X68030とX68000の違いを一言でいえば，あまり使われることになかった信号を廃止したうえで，EXAVEC信号を追加したものということになります。

●図1……X68030で廃止／変更された信号一覧

廃止／変更	信号名	端子番号	備考
廃止	VMA	A34	6800ファミリーLSI接続用信号
	E	B04	
	SELEN	A44	DRAM接続用信号
	CASRDEN	A45	
	CASWRL	A46	
	CASWRU	A47	
	EXPWON	A40	外部からの電源ＯＮ（Compactのみ省略）
	VCC2	A42,A43	バックアップ電源（Compactのみ省略）
変更	EXVPA	A35	EXAVEC（オートベクタ）信号に変更
	IDDIR	B30	拡張スロット側がバスマスタのとき、ハイ・インピーダンスになるよう変更
	FC0	A24	CPUがアクセスする空間を示す　111（CPU空間）は使用不可
	FC1	A25	
	FC2	A26	

2・1 変更された信号

拡張スロットのA35番は信号が変更され，またB30番は信号名や意味もほぼ同等ですが，動作に一部変更が行われました。

2・1・1 EXAVEC（A35）

A35ピンはX68000シリーズではEXVPA信号でした。EXVPA信号はモトローラの8ビットCPU，M6800ファミリーの周辺デバイスを接続するときのための信号です。MC6800はもちろん，そのファミリーデバイスもかなり設計が古く，わざわざ使用することはほとんどないということからX68030では削除され，かわりにEXAVEC信号が割り当てられています。

EXAVEC（Auto VECtor）信号はボード側からベクタを出力せず，デフォルトのベクタを使用するように指示するものです。通常，拡張ボードから割り込みをかけた場合，割り込みアクノリッジサイクルでボードからバスに割り込みベクタを出力しなければなりません。このため，各ボードにはベクタ設定用のレジスタなどが必要でした。X68030では，割り込みアクノリッジサイクルでDTACK信号をアサートするかわりにEXAVEC信号をアサートするとオートベクタとなります。この機能を使うとベクタ設定用のレジスタを省略できるなど，回路を簡略化することができます（DTACKとEXAVECは同時にアサートしないようにしてください）。

オートベクタモードのときのベクタ番号はレベル2が$1A，レベル4は$1Cが使用されます（『Inside X68000』の割り込みの章の図3も参照してください）。

2・1・2 IDDIR（B30）

IDDIR信号は，X68000シリーズではつねに本体側からの出力信号となっていましたが，X68030では拡張ボードがバスマスタとなった場合にはボードからドライブする信号になっています。このため，バスマスタとなるボードの場合，バスを握ったときには必ずIDDIR信号を制御するようにしないと，IDDIR信号を使用している他のボードへのアクセスが不可能になります。

2・1・3 FC0/FC1/FC2（A24/A25/A26）

FC0/FC1/FC2の3本の信号の組み合わせによって，今回のアクセスがスーパーバイザモードかユーザモードか，データか命令であるかといった区別ができるようになっています。X68000ではどの空間へのアクセスでも（本体内部で持っている領域と衝突しないかぎり）利用できましたが，X68030ではバス変換を行っている都合上，拡張バス上の動作としては命令やデ

ータのリード／ライト動作しか示されません。

MC68020以降のCPUでは，コプロセッサとのコミュニケーションを取るときにはFC0/FC1/FC2がすべて‘1’として（「CPU空間」を示します）アクセスを行うようになっています。ところが，X68030では拡張スロットに対するCPU空間アクセスは許されていません。このため，拡張スロットにコプロセッサを取り付けるときには，X68000用の数値演算プロセッサボードのように通常のI/Oデバイスと同じように扱い，コミュニケーション動作をソフトウェアで行う必要があります。

2・2 廃止された信号

X68030では，従来のX68000シリーズの信号のうち，MC6800用の周辺LSIを接続するための信号など，利用されることのなかった信号や，メインメモリが本体内部だけでフル実装できるようになったために不要となったDRAMの制御信号などが省略されています。

また，Compactタイプでは，これらに加えて拡張スロット経由のリモート電源ON/OFF信号や，電源OFF時にも供給され続けるバックアップ電源も省略されています。

これらの信号は純正のオプションボード類では使用していません。拡張スロット用のメモリボードもスロットのDRAM制御信号を使用しない設計となっていますので，廃止されても問題なく使うことができます。

サードパーティ製のボードでもバックアップ電源やリモート電源制御をするものはほとんどありませんし，メモリボードもDRAM関係の信号も純正ボードで使われなかったこともあって使用されているものはほとんどないと思われます。したがって，実際にこれらの信号の廃止によって問題が起こるおそれはないはずです。気になるようでしたら，念のため，廃止された信号ピンからパターンが引かれているかどうかを確認しておくとよいでしょう。

3 拡張スロット信号のDC規格

拡張スロットのDC規格を図２に示します。表中，最大入力電流とあるのは，拡張ボード側で消費することが許されている最大電流を示します。I_{IL}，I_{IH}は，それぞれ拡張スロットのバス上の信号がLレベルのときに拡張ボードからバスに流れ出す電流と，Hレベルのときにバスから拡張ボード側に流れ込む電流を示します。拡張ボード側のレシーバICは，この規格を満たすも

●図2……X68030の拡張スロットのＤＣ規格

信号名	最大入力電流		最小出力電流	
	I_{IL}（mA）	I_{IH}（μA）	I_{OL}（mA）	I_{OH}（mA）
AB1～AB23 DB0～DB15 AS R/W LDS UDS FC0～FC2	−2.0	50	24	−3.0
EXDTACK	−0.4	300	8	—
EXAVEC	—	600＊1	8	—
10MHz $\overline{\text{10MHz}}$ 20MHz	−2.0	500	—	—
HALT	−0.2＊1	250＊1	8	—
EXRESET	−1.2	60	—	—
EXBERR	−0.4	300	8	—
EXPWON	—	200	8	—
IDDIR	−0.8	40	8	−3.0
HSYNC VSYNC	−0.4	60	—	—
INH2	−3.0	300	—	—
DONE	−1.2	120	8	—
DTC	−0.8	60	—	—
EXDREQ	—	120	8	—
EXDACK	−0.8	60	—	—
EXPCL	−0.8	60	8	−1.0
EXOWN	−0.8	120	8	—
IREQ2-1,IREQ2-2 IREQ4-1,IREQ4-2	—	120	8	—
EXNMI	—	120	—	—
IACK2-1,IACK2-2 IACK4-1,IACK4-2	−0.8	120	—	—
BR-1,BR-2	—	120	8	—
BG-1,BG-2	−0.8	120	—	—
BGACK	−0.8	60	8	−1.0

＊1：全スロットの合計値。

のを使用しなければなりません。

最小出力電流は，拡張ボードがその信号を使用するときに最低限どれだけの電流を流すことができなければならないかを示すものです。拡張ボード側のドライバICはこの規格を満たすものを使用する必要があります。

表中の数値の符号は，バスから拡張ボードに流れ込む向きのとき正(＋)，拡張ボードからバスに向かって流れるときを負(−)としています。たとえば，I_{IL}なら，信号がLowレベルのときにボードからバスに向かって流れる電流を示しますので，符号がマイナスになります。また，I_{OL}は拡張ボードがその信号をLowレベルにドライブしたとき，バスから拡張ボードに向かっ

て流れる電流ですから符号はプラスになるわけです。

4 拡張スロットの電流容量

拡張スロットで消費することのできる電流容量を図3に示します。IDDIRは，バスマスタになるボードがあった場合，ボード側からドライブする必要があるため，X68030で追加された仕様です。

●図3……拡張スロットの電流容量（1スロットあたり）

電源種別	電流容量
VCC1（＋5V）	600mA
VCC2（＋5V）	30mA
VCC3（＋12V）	30mA
VCC4（－12V）	30mA

VCC1は，通常の＋5V電源，VCC2は本体正面の電源スイッチがOFFになっていても供給され続ける＋5Vの電源で，本体の電源LEDやリアルタイムクロック，キーボードなどはこの電源で動作しています。

CZ-300C/CZ-310C (X68030 Compact) では，VCC2はスロットに供給されていませんので注意してください。

5 拡張スロットの動作タイミング

X68000の拡張スロットの信号はほとんどCPUと直結されたものであり，CPUの資料を見ればクロックとの関係がわかったのですが，X68030ではCPUの動作がMC68000とはまったく異なるため，MC68000に似せた信号をASICで作成しています。ASICを設計するときには，内部回路はすべてクロックに同期して動くように設計するのが基本ですから，当然X68030の拡張スロットタイミングもクロックに同期して動いているはずなのですが，残念ながら，シャープからはクロックを基準とした動作タイミングは公開されず，動作タイミングも各信号の変化

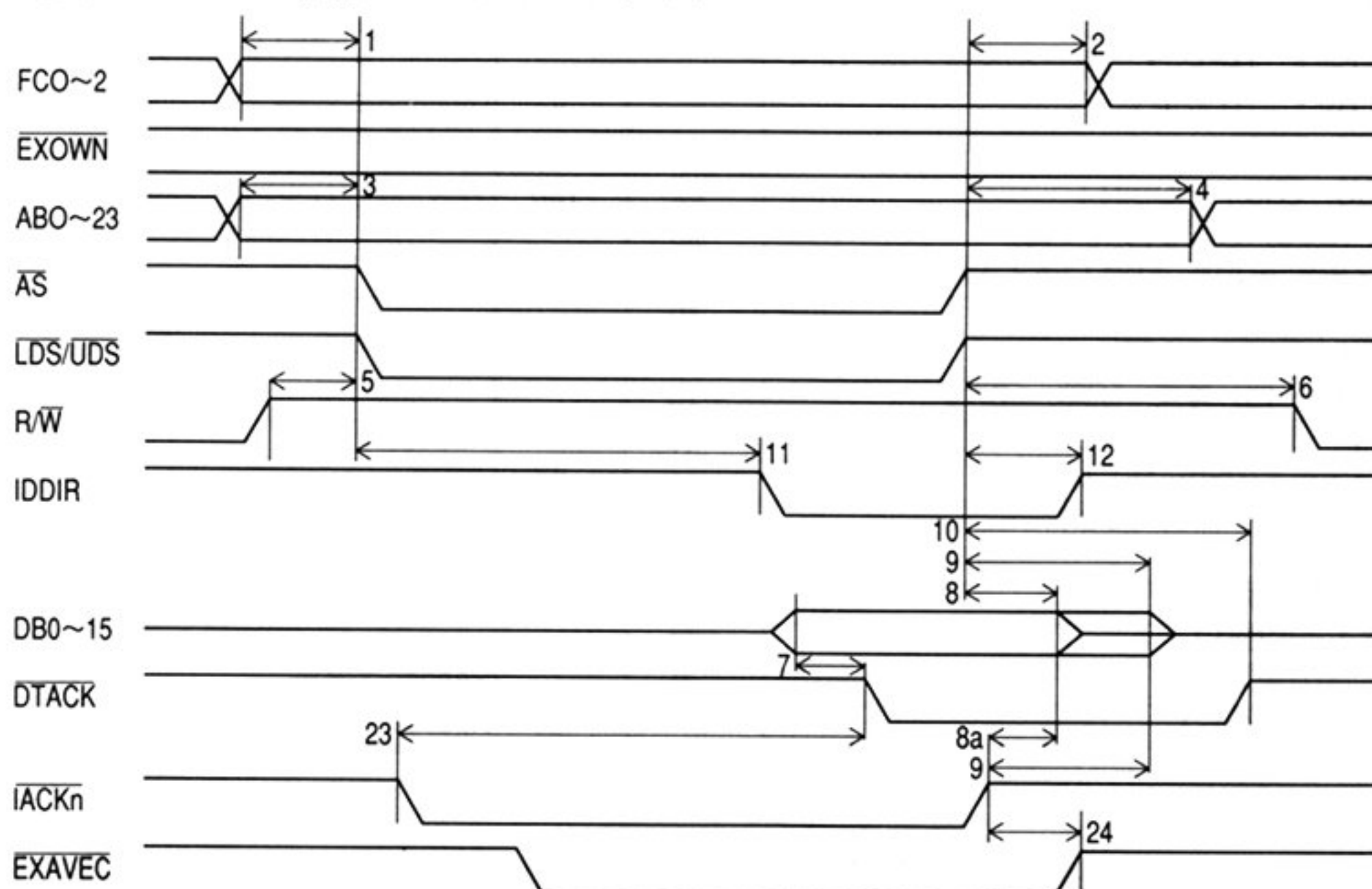
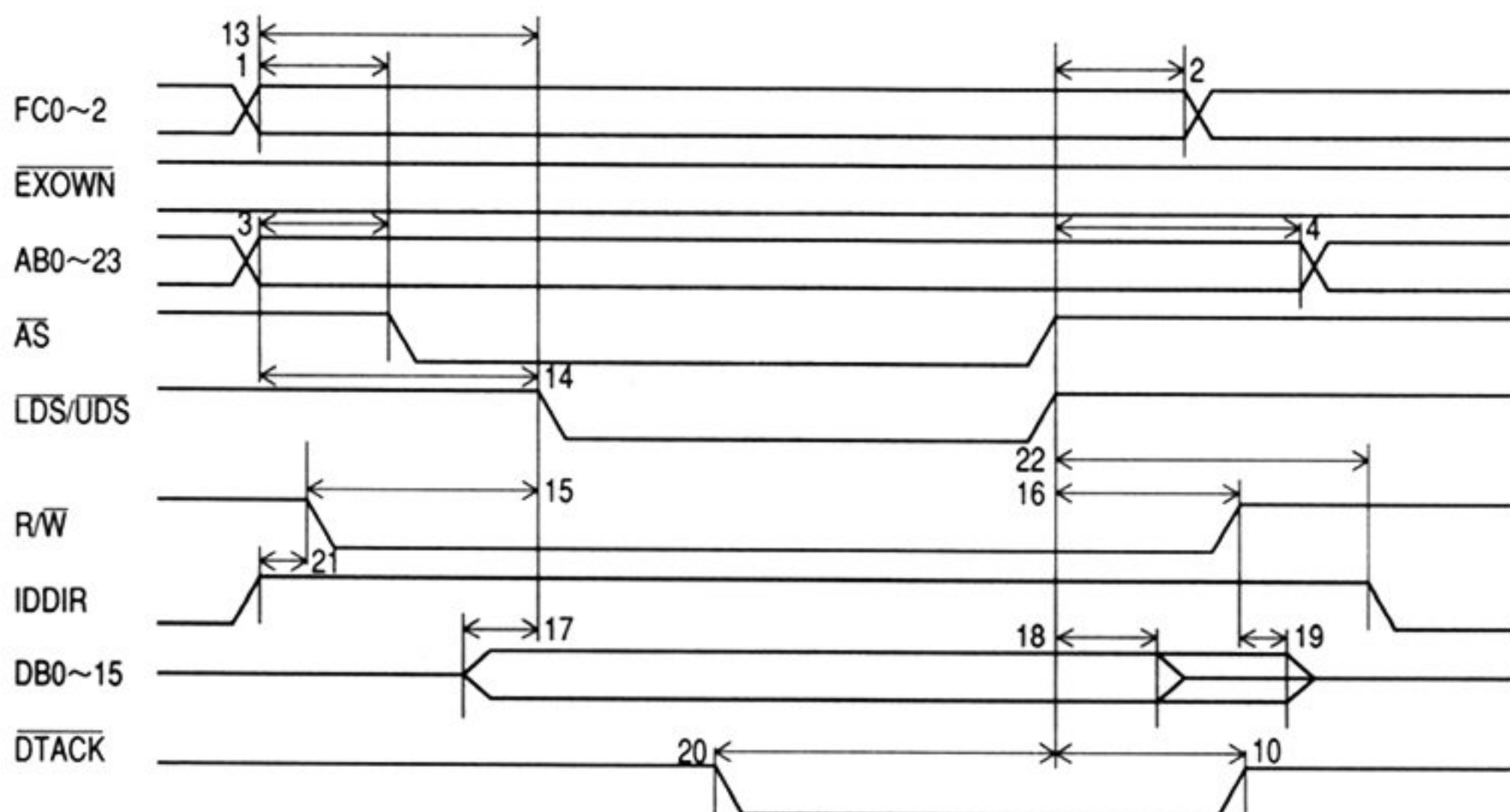
点からの規定になっています。このあたりの事情はX68000の拡張スロットのときと同じです。

ただし，拡張ボード側がバスマスタとなって動作する場合には10MHzのクロックに同期して動くことが要求されています。これは，拡張ボードがバスマスタになっているときはゲートアレイが拡張スロットの各信号をクロックに同期して取り込んでいるためと考えられます。ただ，ゲートアレイの内部動作はかなり高速ですからクリティカルになる時間はさほど長くありません。クロック同期を気にせずに設計してもよほど運が悪くないかぎり問題が起こることはないはずですが，極力クロックに同期した設計にするようにしてください。

5・1 タイミングの詳細とX68000との違い

図4にX68030の拡張スロットのリードサイクルのタイミング図を，図5にライトサイクルのタイミング図を示します。IACKnとEXAVECTは割り込みの応答サイクル時に動く信号であり，通常のメモリアクセス時に動くものではありませんが，タイミング上は通常のメモリリードサイクルと同じような動作になるのでいっしょになっています。

図中，番号の振ってある部分の具体的な値は図6にまとめてあります。表中，太字になっているのは，X68000の拡張スロットタイミングと値が違うものや，新しく規定されたものを示し

●図4……X68030拡張スロットリードサイクル

●図5……X68030拡張スロットライトサイクル

●図6……X68030拡張スロットタイミング値

番号	項　　目	遅延時間（nsec）		備　　考
		MIN	MAX	
1	FC確定→AS（リード時はUDS、LSDも）↓	10		
2	AS、DS↑→FC無効	10		
3	アドレス確定→AS（リード時はUDS、LDSも）↓	10		
4	AS、DS↑→AS、UDS、LDS↓	10		
5	R/W↑→AS、UDS、LDS↓	30		
6	AS、UDS、LDS↑→R/W↓	80		
7	データ確定→DTACK↓（リード時）	0		DTACKはデータ確定後にアサートすること
8	UDS、LDS↑からのデータホールド時間	0		ベクタ割り込みアクノリッジサイクルではいずれか一方を満足すればよい
8 a	IACKn↑からのデータホールド時間	0		
9	UDS、LDS↑，IACKn↑→データ(DB0~16) Hi-Z		80	
10	UDS、LDS↑→DTACK↑		**80**	X68000では130
11	AS、UDS、LDS↓→IDDIR↓		150	
12	AS、UDS、LDS↑→IDDIR↑	**25**	**180**	X68000では45/150
13	FC確定→UDS、LDS↓(ライト時)	90		
14	アドレス確定→UDS、LDS↓(ライト時)	45		
15	R/W↓→UDS、LDS↓	40		
16	AS、UDS、LDS↑→R/W↑	10		
17	データ(DB0~15) 確定→UDS、LDS↓	−10		
18	DS↑→データ無効	**−10**		X68000では10
19	R/W↑→データ(DB0~15) Hi-Z		35	
20	DTACK↓→AS、UDS、LDS↑	**80**	**360**	DTACKはデータ取り込み後にアサートすること X68000では300
21	IDDIR↑→AS、R/W↓	10		
22	AS、DS↑→IDDIR↓	180		
23	IACKn↓→DTACK↓	**0**		X68030で追加
24	IACKn↑→EXAVEC↑		**80**	X68030で追加

ます。

X68000とX68030の大きな違いは，割り込みアクノリッジサイクルのときのIACKn信号のタイミング規定が変わったことと新しく追加されたEXAVEC信号についての規定が追加されたことです。

IACKn信号は，X68000のときには変化するところがUDS/LDSの変化点から規定されていましたが，X68030ではこの時間規定は削除され，かわりにIACKnが'L'になってからDTACKを'L'にするまでの時間が規定されました。

EXAVEC信号はX68000にはなかった信号です。EXAVEC信号は，割り込みアクノリッジサイクルで，ボード側からDTACK信号を返すかわりにEXAVEC信号を返すことでオートベクタ動作になるものです。DTACK信号はLDS/UDSの立ち上がりをみて引き上げますが，EXAVEC信号はIACKnの立ち上がりをみて引き上げるように規定されています。

X68030の動作タイミング実測結果

X68030は，CPUの動作がX68000とまったく異なるにもかかわらず，周辺デバイス関係はX68000と同一としているため，動作タイミングが複雑になっています。ここでは，X68030の主要タイミングを実測した結果を整理してみました。

1 主要デバイスへのアクセスタイミング

X68000では基本的にCPUバスがそのまま周辺デバイスに供給されていましたが，X68030ではメインメモリや数値演算プロセッサは32ビットのローカルバス，VRAMやI/Oデバイスは16ビットバス，拡張スロットはこれらとは別の10MHzの16ビットバス（MC68000準拠のタイミング）で動作するなど，バス構成がやや複雑になっています。また，X68030では，高速になったCPUの動作をできるだけ妨げないようにするため，メインメモリを含め周辺デバイスのアクセスタイミングが細かく制御されているほか，システムポートによってメインメモリやROMのアクセス速度が調整できるようになっています。しかし，これらの具体的なタイミング数値は公開されていません。

実際にプログラムを組むうえでメモリアクセスやI/Oへのアクセス時間が予測できないというのは非常に不便なので，主だったデバイスの実際のアクセスタイミングやウェイト数などを実測してみました。測定したタイミングは以下のとおりです。

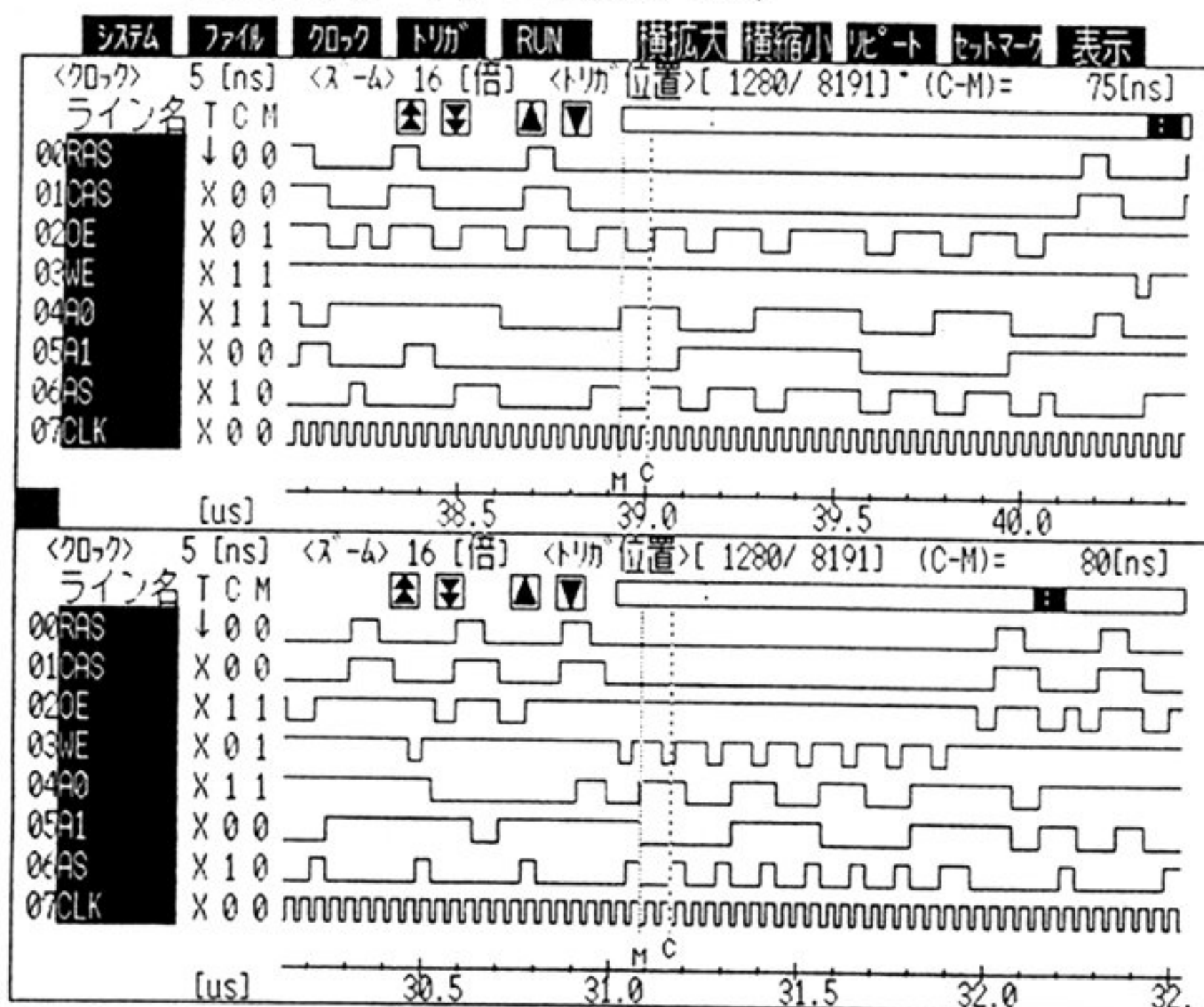
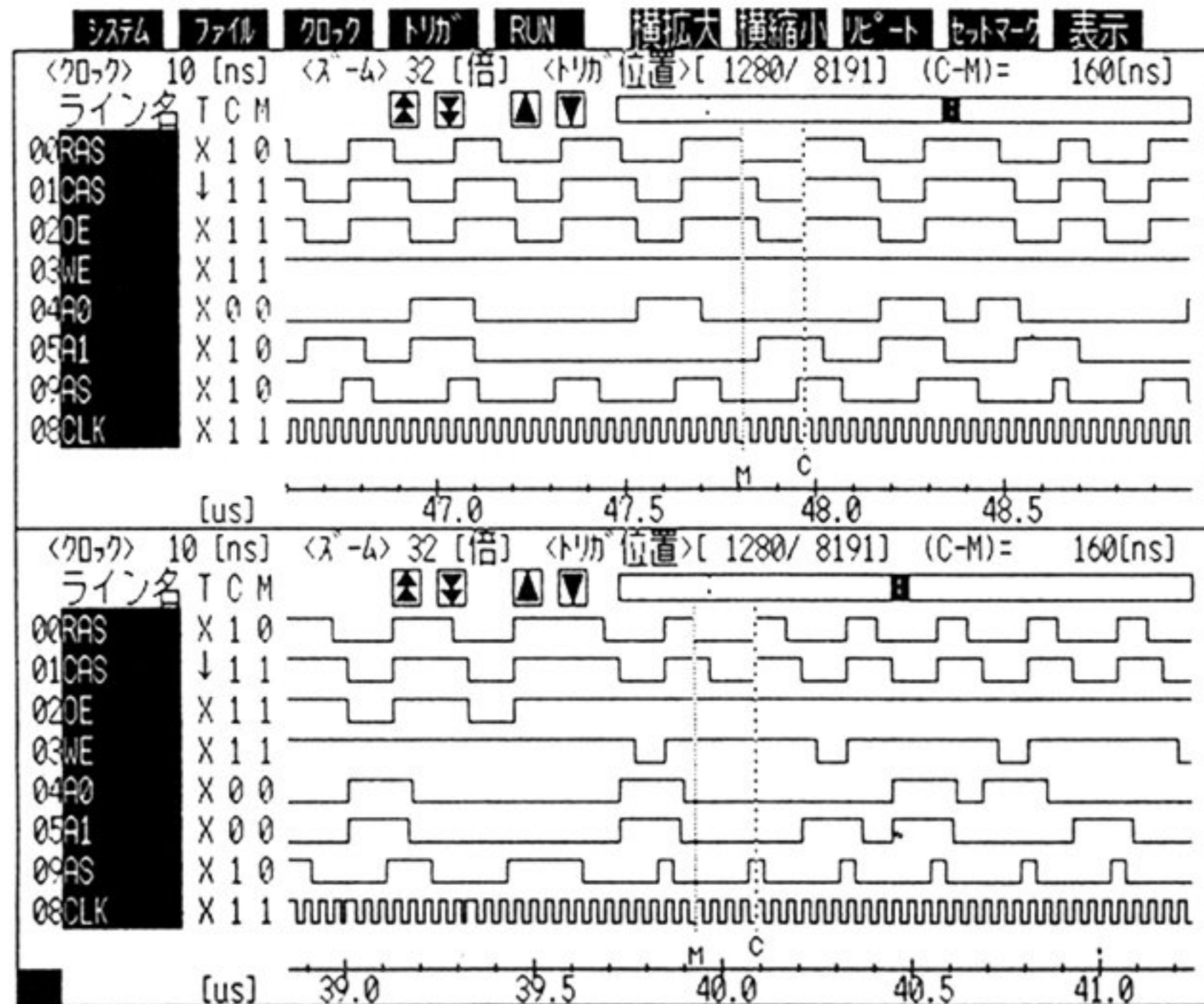
・DRAMアクセスタイミング

・ROMアクセスタイミング
・拡張スロットアクセスタイミング
・DMAによるデータ転送動作タイミング
・周辺デバイスアクセス時のウェイト数

2 DRAMアクセスタイミング

X68030では，メモリアクセスを高速化するため，DRAMの高速アクセスモードの１つであるスタティックカラムモードを利用しています。この様子を図１に示します。図の上半分がスタティックカラムでのリードサイクル，下半分がライト動作をしているときのタイミングです。信号名のASとCLKは，それぞれCPUのアドレスストローブピン，クロック入力ピンの波形を示し，それ以外はDRAMの信号ピンの波形を示します。

RASとCSが'L'になったままOEやWEが連続しているのが，スタティックカラムモードで

●図1……スタティックカラムモードでのDRAMアクセス
(上半分がリード、下半分がライト)

●図2……ウェイト数に$1を設定したときのアクセスタイミング
（カーソルはDRAMのアクセスサイクル幅）

動作しているところです。スタティックカラムモードで動作する場合，最初のアクセスは7クロック（4ウェイト）かかっていますが，2回目以降のアクセスはノーウェイトで動作していることがわかります。

スタティックカラムモードが使用されるのは，システムポート#5の下位4ビットが$0になっているときだけです。$0以外になっていると，スタティックカラムモードでの動作は行われなくなり，通常アクセスだけになります。図2に$1を設定したときのアクセスタイミングを示します。値が$1のとき，DRAMアクセスのサイクルタイム(RASが‘L’になってから‘H’に戻るまで)は160nSあります。CPUのアクセスサイクルはAS信号の動きからみて6クロックサイクル(240nS) です。68030の非同期バスサイクルは3クロックサイクルですから，3ウェイト入っていることがわかります。

さらに，システムポートの下位4ビットを増減してみると，設定値が1増えるたびに1ウェイト (40nS：1クロック分) ずつ追加されることがわかりました。このことから，システムポート#5の下位4ビットへの設定値とアクセス時のウェイト数の関係は図3のようになります。

●図3……システムポート#5のウェイト設定値とDRAM/ROMアクセスウェイト数の関係

設定値	DRAMアクセスウェイト	ROMアクセスウェイト	備　考
$0	0(*)	2	
$1	3	3	
$2	4	4	
$3	5	5	
$4	6	6	16MHz相当モード
$5	7	7	
$6	8	8	
$7	9	9	
$8	10	10	
$9	11	11	
$A	12	12	10MHz相当モード
$B	13	13	
$C	14	14	
$D	15	15	
$E	16	16	
$F	17	17	

*スタティックカラムモード動作中の値。ページ間をまたぐときは4ウェイト。

2・1 DRAMのアクセスモード

2・1・1 DRAMの構造

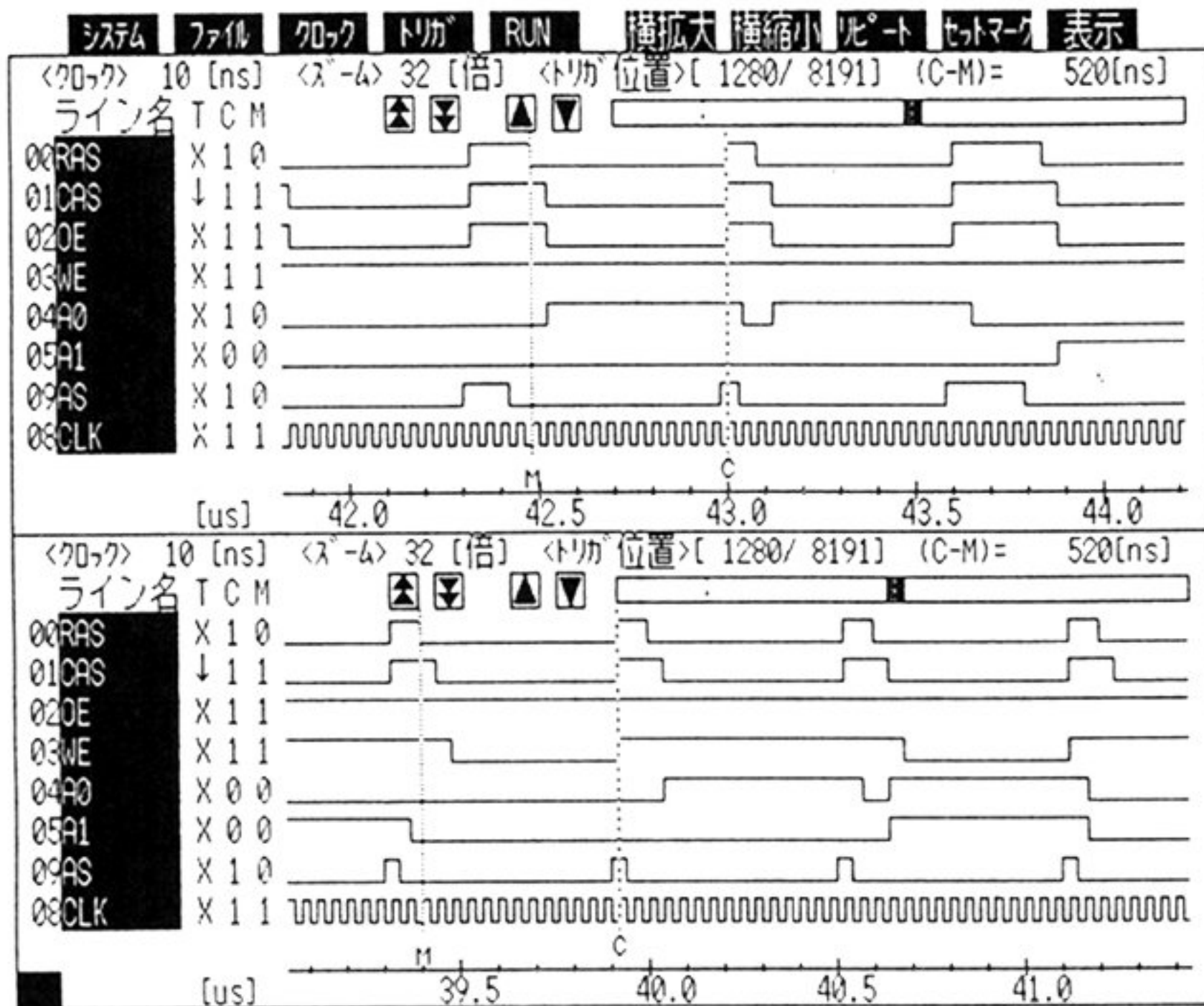
DRAMのなかのおおざっぱな構造は126ページの図6のようになっています。DRAMはアドレスピン数の削減のため，アドレスをロウアドレス（Row Address：行アドレス）とカ

COLUMN

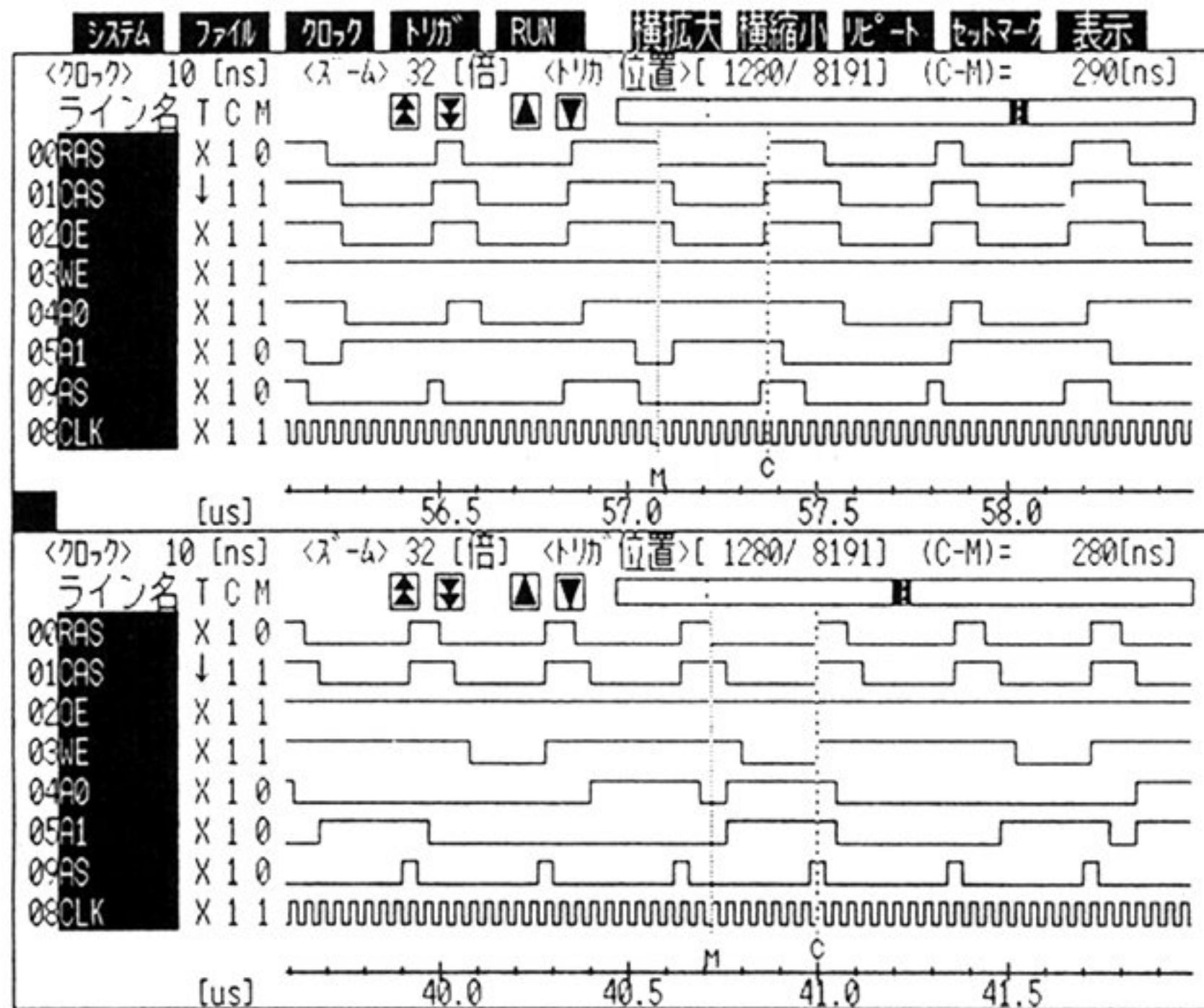
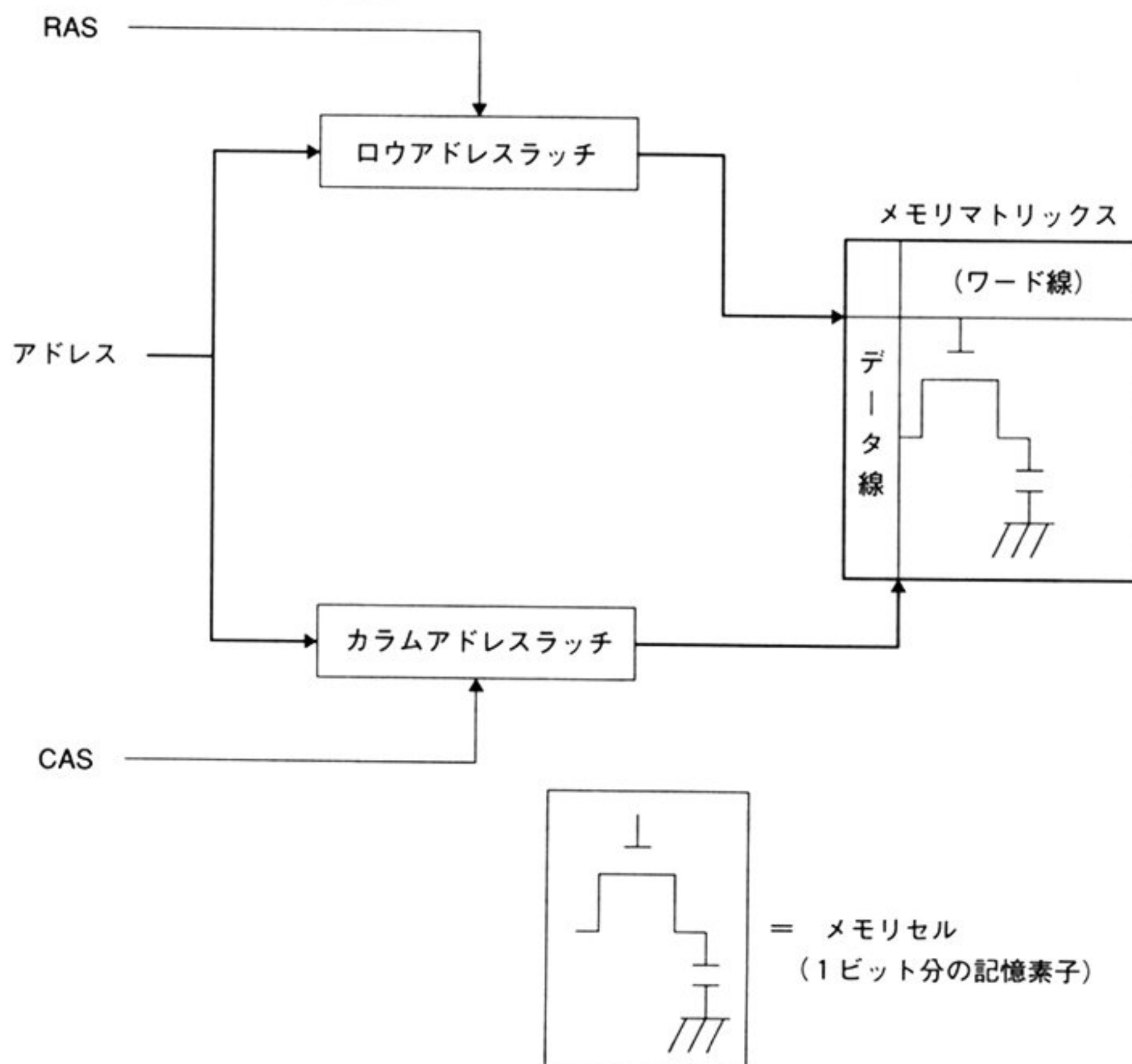
X68030の10MHz/16MHzモード

X68030を[XF3]キーや[XF4]キーを押しながら立ち上げると，10MHz，16MHzのX68000に近い状態で立ち上がります。このときの処理速度の引き下げはクロックを落とすのではなく，CPUのキャッシュをOFFにしたうえでシステムポート#5を使って処理速度を引き下げることで行われています。

実際に[XF3]キーや[XF4]キーを押しながら立ち上げた状態でDRAMへのアクセスタイミングを実測した結果を図4と図5に示します。DRAMアクセスのサイクルタイムがそれぞれ520nS，280nSとなっています。ここから，10MHzモードのときには10(520＝160＋9×40)を，16MHzモード時には4(280＝160＋3×40)を設定していることがわかります。

●図4…… [XF3]キーを押しながら立ち上げたときのDRAMアクセス

●図5…… [XF4]キーを押しながら立ち上げたときのDRAMアクセス

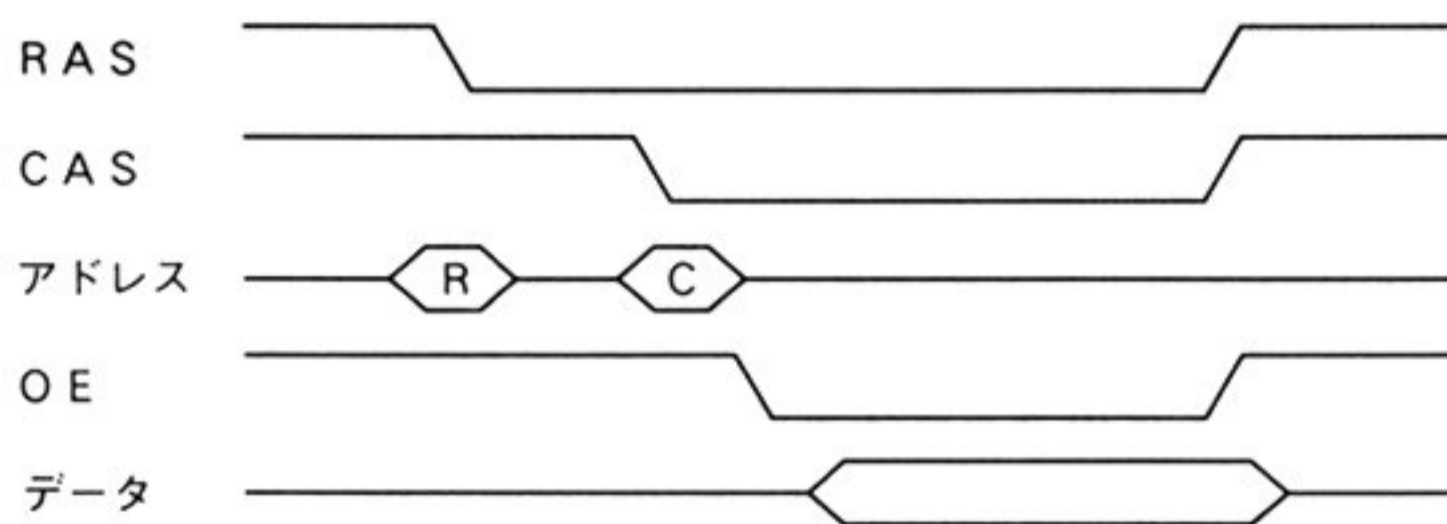
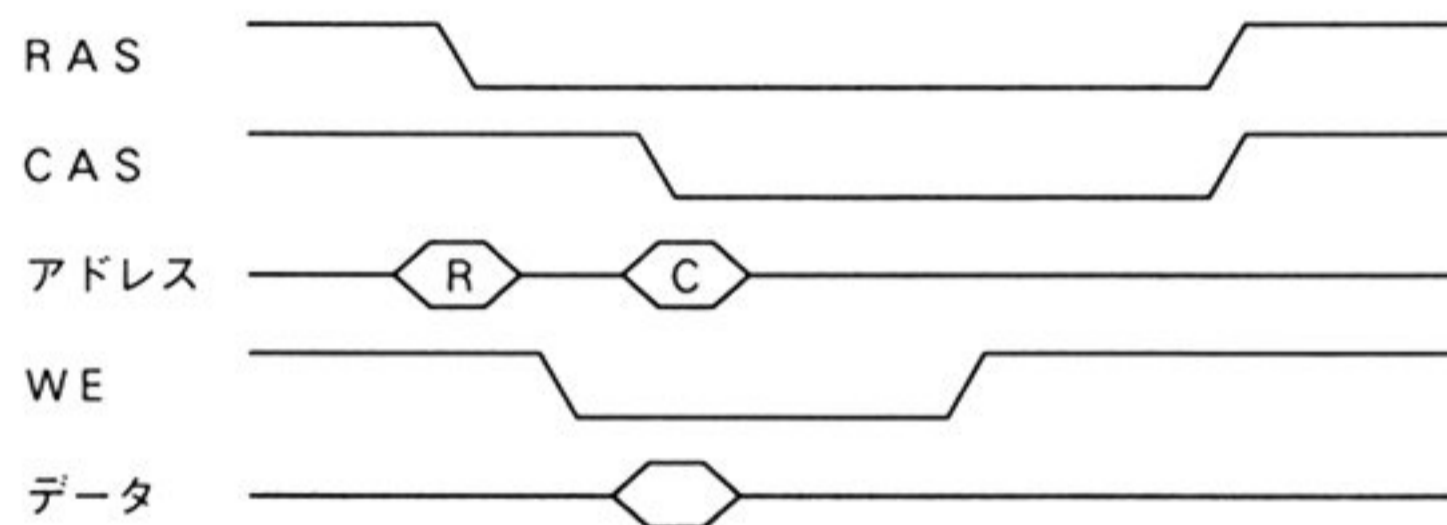
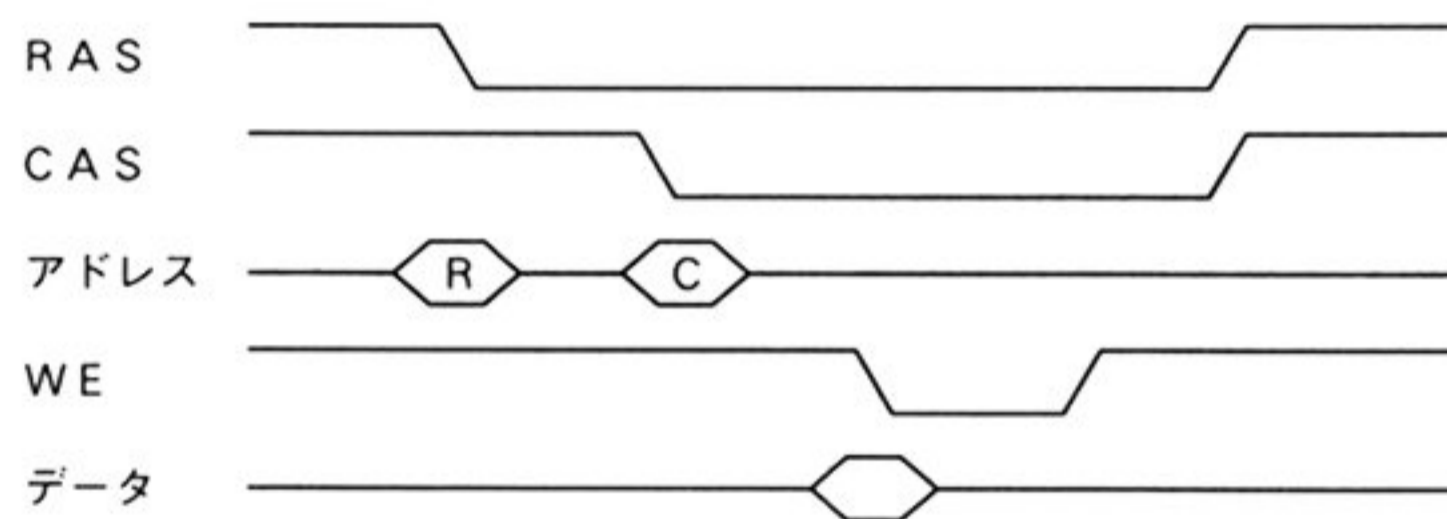
●図6……DRAMの内部構造

RAS
ロウアドレスラッチ
メモリマトリックス
(ワード線)
アドレス
データ線
カラムアドレスラッチ
CAS
= メモリセル
(1ビット分の記憶素子)

ラムアドレス(Colum Address：列アドレス) に分けて与えます。この2回に分けて与えられるアドレスを取り込むタイミングを与える信号として，RAS (Row Address Strobe) とCAS (Colum Address Strobe) があります。

2・1・2 DRAMの基本的なアクセス方法

DRAMの基本的な使い方を図7に示します。まず，ロウアドレスを与えて，どのワード線を使うかを指定し，次にカラムアドレスを与えて選ばれたワード線上の，どのメモリセルをアクセスするかを決定します。

ライト動作は，アクセス中にWE (Write Enable) 信号を‘L’にすることで行います。WEを‘L’に落とすタイミングをCASよりも前にするか後にするかによって，アーリーライトとディレイドライトの2種類に分かれます。ウェイトが入ったときのタイミング図を見ると，X68030ではディレイドライトが使われていることがわかります。

●図7……DRAMの基本アクセス

リード動作はWEを'H'のままにするとリードモードになるのが一般的ですが，X68030に使用されているDRAM(HM514402)は，データを出力するか否かを制御するOE(Output Enable)端子を持っているため，OEの制御が行われます。スタティックカラムモードでのリード時はOEを'L'にしたままでもよいのですが，X68030では1回のアクセスのたびに'H'に戻しています。

2·1·3 高速アクセスモード

DRAMはスタティックRAMに比べるとローコストで大容量のものを実現できるのですが，一方，アクセス時間やサイクル時間はどうしても長くなりがちです。これを改善するためにロウアドレスが変化しない場合にはロウアドレスを再ラッチするのを省略してアクセスする方法が考えられました。これを「高速アクセスモード」と呼びます。代表的な高速アクセスモードとしてはページモード，ニブルモード，スタティックカラムモード，高速ページモードの4種類があります。これらのモードのうち，どれが使えるかはDRAMごとに決まっています。同時に2つのモードが使えるDRAMはありませんので，DRAMを選択するときには注意が必要です。高速アクセスモードのうち，現在，主流となっているのは高速ページモードです。市販のDRAMモジュールなどでも使われているのは高速ページモードのものばかりで，スタティックカラムモードのものはほとんどありません。拡張メモリボードの自作などをされる方は気をつけてください。

各高速動作モードの動作の概略を図8に示します。なお，以下の説明ではDRAMのデータ長は1ビットであるとします。実際にX68030で使用されているDRAMは1チップあたり4ビットのものですが，これは単に同じものが4つ押し込まれていると考えてください。

ページモード

ページモードは，RASを'L'に固定した状態でCASを使って次々にカラムアドレスを与えることで，同一ワード線上のセルを選択して読み出す方法です。最も標準的な高速アクセス方法でしたが，現在では後で説明する高速ページモードにとって代わられています。

ニブルモード

DRAMの内部で出力端子のそばに4ビット分のラッチを設けておいて，カラムアドレスが与えられたときに連続する4ビット分のデータをラッチしておき，CASによってこれらのデータを順次読み出す方法です。4ビット分しか連続アクセスできませんが，ページモードよりも高速にアクセスできる方法です。

スタティックカラムモード

X68030で使われたのがこの方法です。高速スタティックRAMとDRAMを融合させたような構造になっていて，CASを'L'にしたままアドレスを変化させるだけでアクセスすることができます。

スタティックカラムモードのDRAMではカラムアドレスはラッチされないので，CASとい

●図8……DRAMの高速アクセスモード

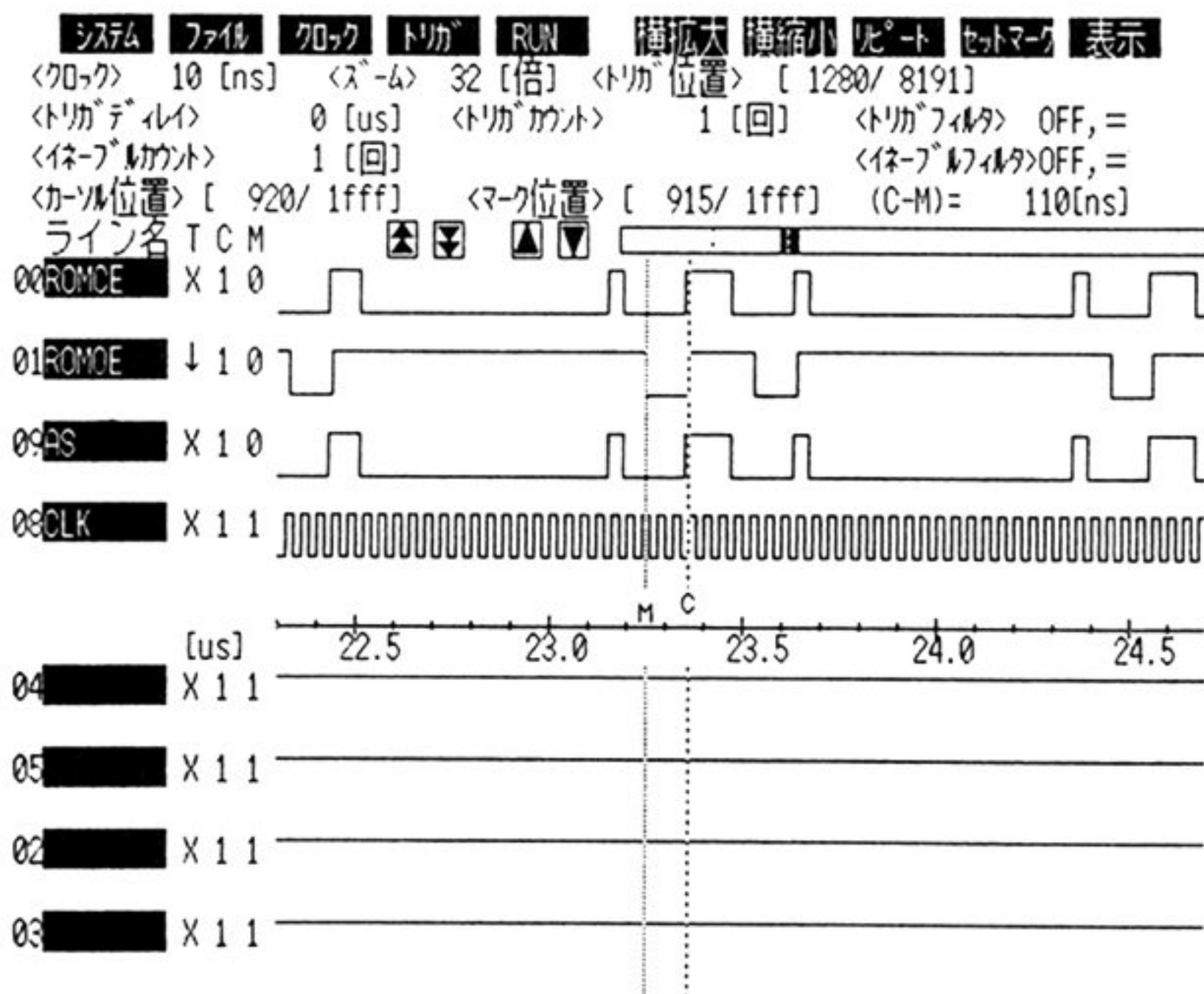
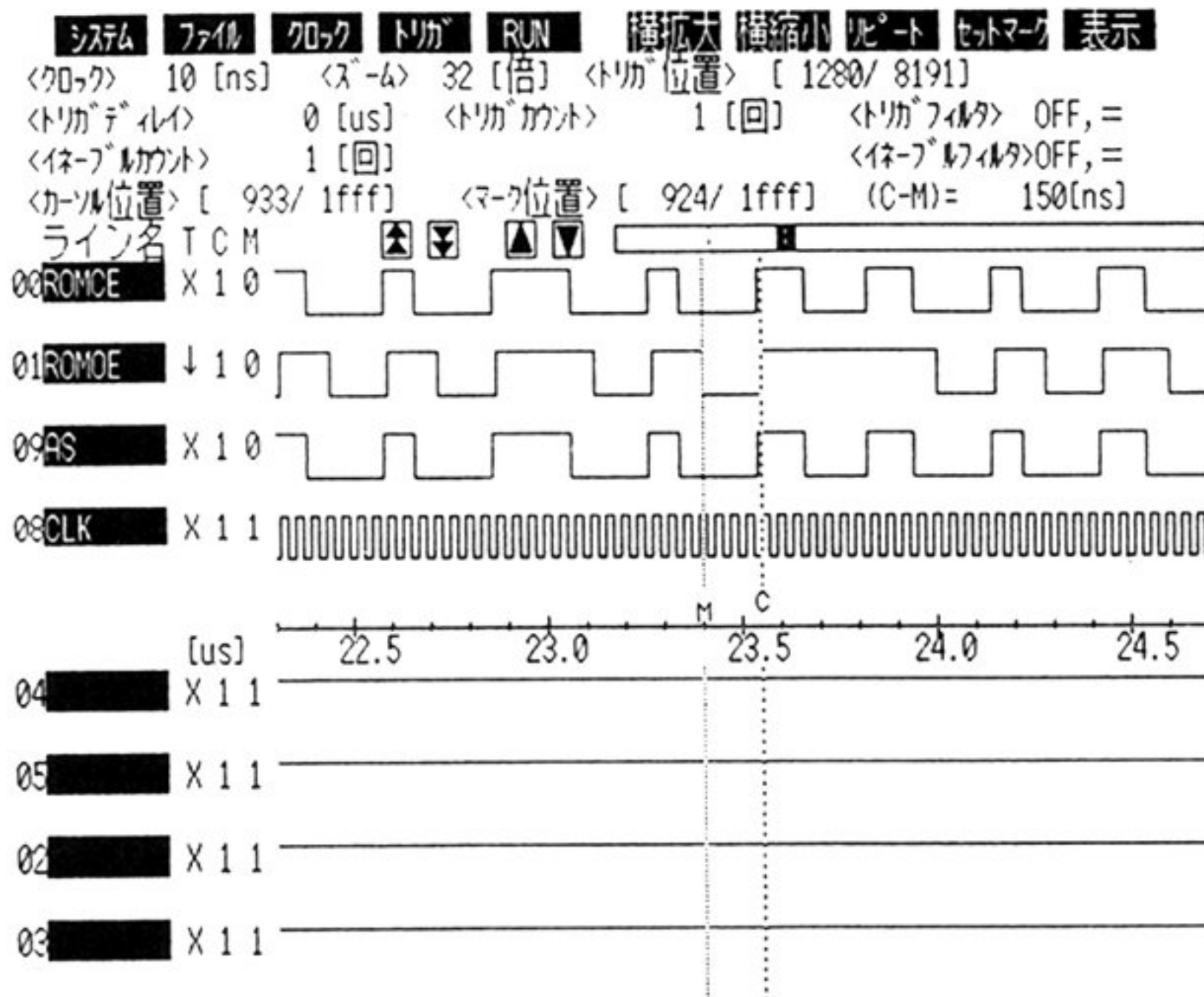
う呼び方はふさわしくないためか，メモリメーカのカタログなどでは信号名をCS（Chip Select）としています。

高速ページモード

スタティックカラムモードのDRAMを，カラムアドレスをラッチするようにしたものです。RASを'L'に固定したままCASによってカラムアドレスを次々に与えるという方法になります。速度が速いだけで，使い方そのものはページモードと変わりません。

3 ROMアクセスタイミング

図3にも示したように，ROMへのアクセス時のウェイト数もシステムポート#5の上位4ビット（ビット4～7）で変更することが可能となっています。この4ビットを$0としたときの波形を図9に，$1を設定したときの波形を図10に示します。CEとOEが両方とも'L'になるサイクルがROMをアクセスするサイクルです。ROMアクセス時のサイクル時間は図から200nS（5クロックサイクル）と読み取れます。CPUの非同期バスサイクルは3サイクルですから2ウェイト入っていることになります。システムポート#5の設定値とウェイト数の関係はDRAM

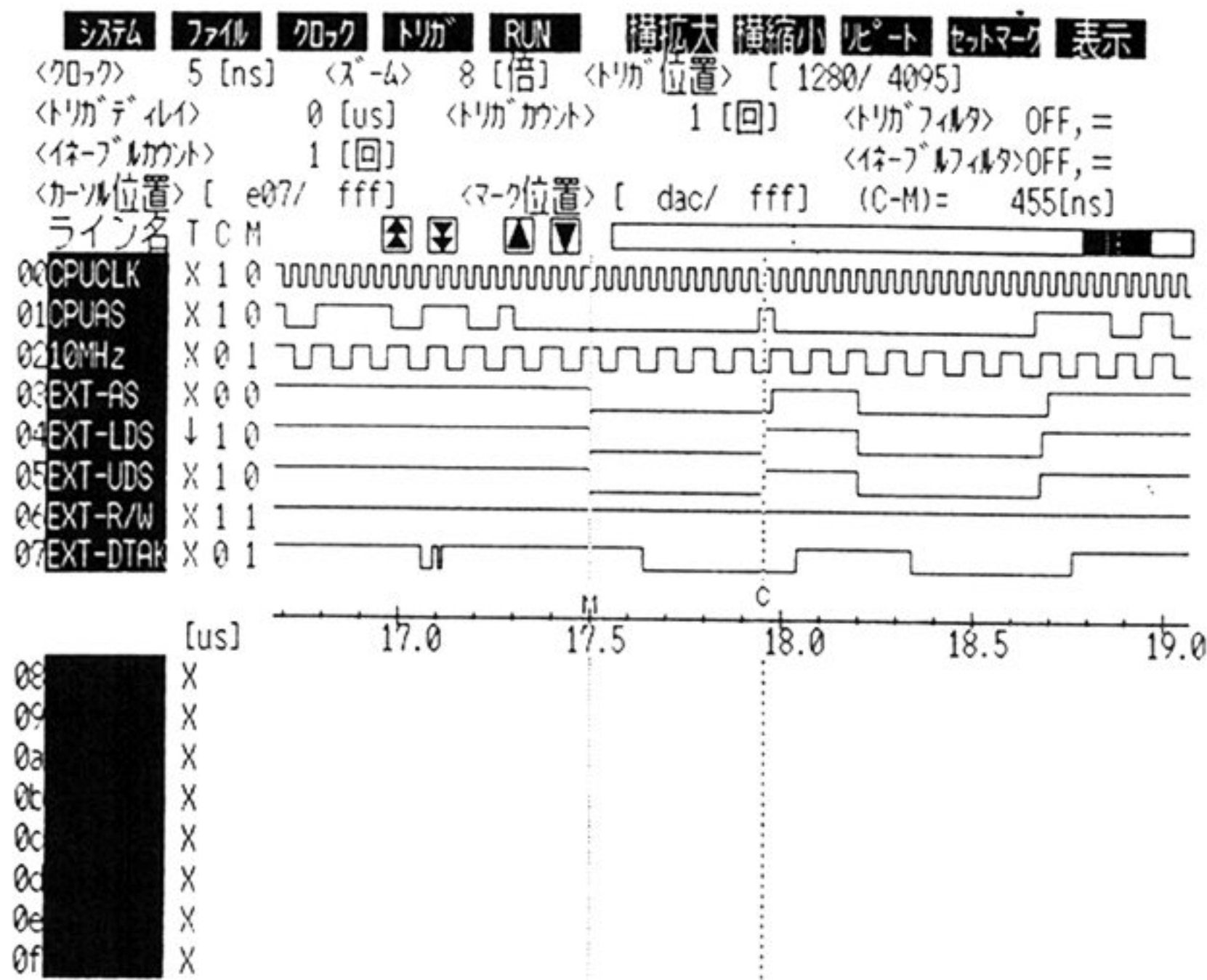
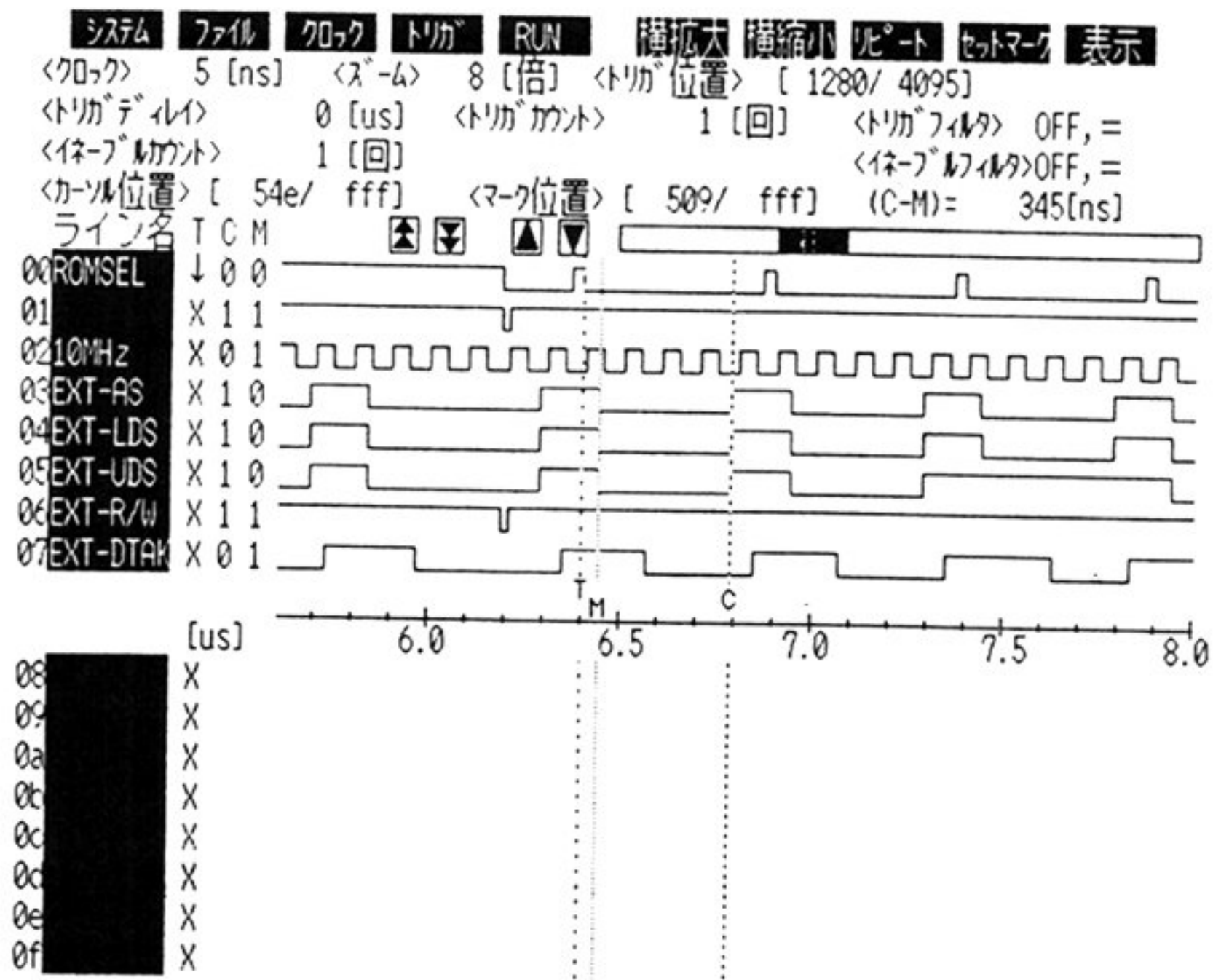
●図9……ウェイト数0のときのROMアクセス
（カーソルはROMのOE信号幅）

●図10……ウェイト数1のときのROMアクセス
(カーソルはROMのOE信号幅)

のときと同様で，設定値が1増えるたびに40nS（1クロックサイクル）だけ延長されます。

[XF3]キーや[XF4]キーを押しながら立ち上げたときの波形も実測してみましたが，ROMへのアクセスタイミングは変化しませんでした。

4 拡張スロットアクセスタイミング

拡張スロットにSCSIインターフェースボードを入れ，ボード上のROMをアクセスしたときのタイミングを図11に示します。X68030にSCSIボードを入れても内部SCSIとのからみで起動することはできませんが，アクセス時の波形を取るには手ごろだったので使用しました。CPUCLK，CPUASは，それぞれCPUのクロック入力とAS（アドレスストローブ）信号（出力）です。10MHzやEXT-ASなどの信号はすべて拡張スロット上の信号です（DTACK信号は文字数が多すぎるのでEXT-DTAKと短縮しています）。

なお，X68030で拡張スロットの信号を見るときには，拡張ボードを入れるか，拡張スロットにアクセスしたときにDTACKを返すような回路をつけて，拡張スロットの空間にアクセスする必要があります。また，X68000の場合には，拡張スロットに出ているのはCPUの信号そのも

●図11……拡張スロットアクセス
（カーソルはUDS/LDSの幅）

●図12……X68000での拡張スロットアクセス
（カーソルはUDS/LDSの幅）

のでしたので，CPUが内部のI/Oなどを操作しているときにもASやLDSなどの信号が動いていましたが，X68030の場合には拡張スロットにアクセスしたときしか動作しません。波形を観測するときには気をつけてください。

さて，拡張バスに目を移すと，CPUからのアクセスが10MHzのクロックに同期した信号に変換されている様子がよくわかります。比較のためにX68000で測定した結果が図12です。かなりよく似たタイミングが作られていますが，AS，UDS，LDSなどの変化とクロックの位相関係が異なっています。ASの立ち上がりとUDS/LDSの立ち上がりがX68000の場合にはほぼ同時であったのが，X68030ではASが少し遅れて立ち上がることなど，若干違う点があることがわかります。

また，拡張バス上の動作がX68000のときには5サイクル（1ウェイト）で完了していたのが，X68030では6サイクルと1サイクル増えています。MC68000の場合，DTACKのセンスはS4の終わり，つまり，ASが'L'になった後の2回目のクロックの立ち下がりでサンプリングされます。DTACKはこの時点ではまだ'H'のままなので，1ウェイト入るわけです。

X68030の場合もDTACKが'L'になるタイミングは大差ありませんので，DTACKのセンスを同じような方法で行っていれば1ウェイトしか入らないはずですが，現実には2ウェイト入ったような動作をしています。

X68030の拡張スロットにボードを入れた場合，拡張スロットへのアクセス頻度によってはX68000でアクセスしたときよりも遅くなる可能性があることがわかります。

5 DMAによるデータ転送動作タイミング

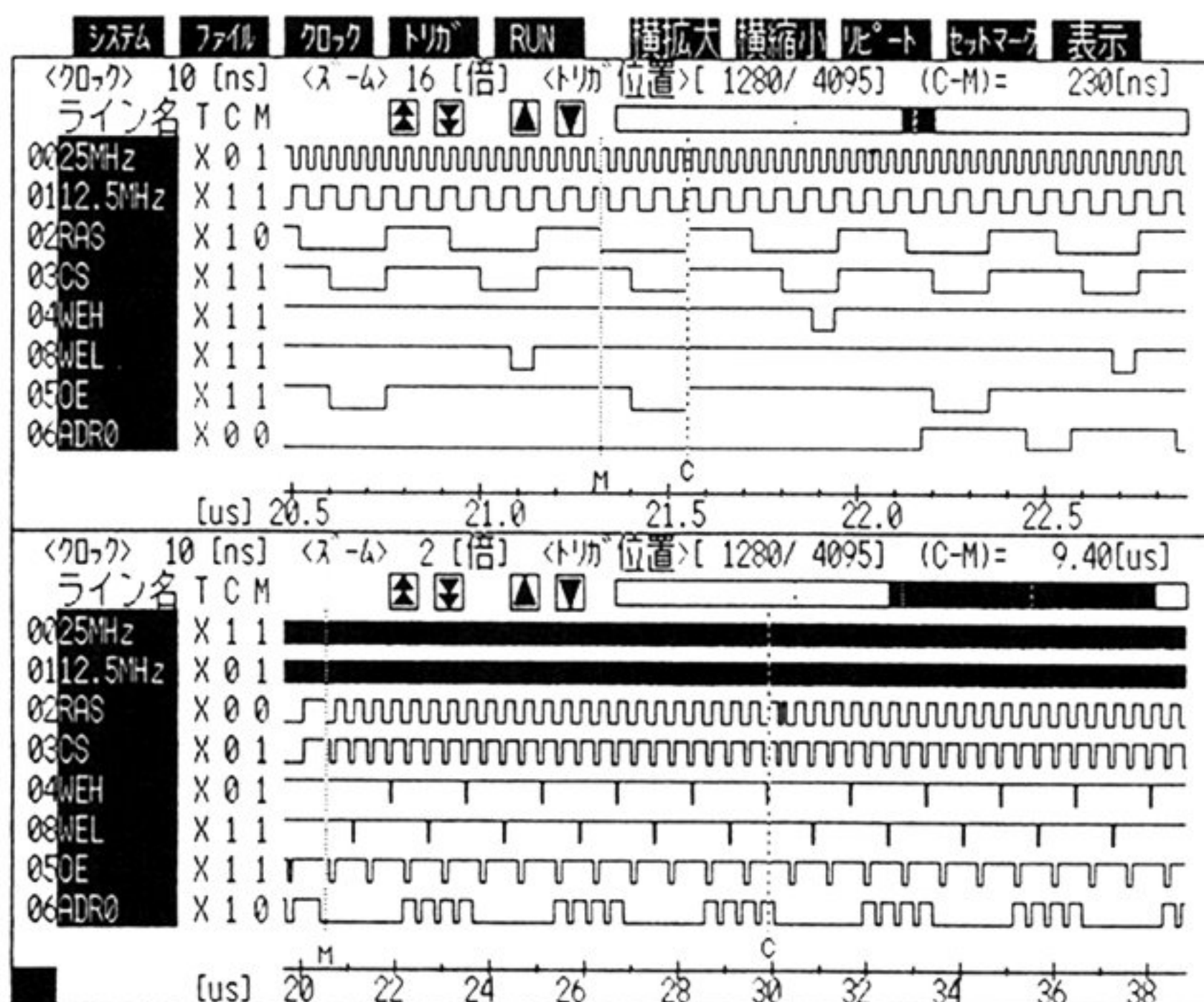
DMAを使ってメインメモリ間のデータ転送を行ったときのメモリへのアクセスタイミングを図13に示します。図の上半分は，下半分の波形のうち，最初の転送の部分を拡大したものです。DMAの転送モードを最大速度に設定していますので，転送が開始されると終了するまでDMAがバスを使い続けます。

図中，WEH，WELは，それぞれ上位16ビット，下位16ビットへのライト信号です。図の下半分を見ると，DMAは16ビットずつリード，ライトを行っていることがわかります。メインメモリは32ビット幅ですが，DMAのデータバスは16ビットしかないためです。

次に上半分を見ると，メモリのアクセスサイクルは400nS，1回の転送サイクルは800nSであることがわかります。MC68000のメモリアクセスサイクルはノーウェイトの場合4サイクルですから，転送速度は10MHzのMC68000相当です。回路図を見ると，DMAコントローラのクロ

●図13……DMA転送時の波形

上半分が拡大図（カーソルはRASの立ち下がりからRASの立ち上がりまで）

下半分は縮小図（Cカーソルはリフレッシュサイクル）

ックとしては12.5MHzが与えられているのですが，実際の転送動作を見るかぎり，クロックを引き上げた効果はあまり表れていません。

転送動作が10MHzの68000相当であるうえ，シングルアドレスモードが使えないため，X68030でのDMAによる転送よりもCPUによる転送のほうがはるかに高速になっています。

また，DMA転送動作とは直接関係ありませんが，図の下半分のなかで1ヵ所だけ，少し波形が乱れたように見える部分があります（Cカーソルの近辺）。この部分をよく見ると，CSがRASより先に'L'になっていますので，これはDRAMアクセスではなく，CSビフォーRASリフレッシュによるDRAMのリフレッシュサイクルです。

周辺デバイスアクセス時のウェイト数

X68030のCPU, MC68EC030は32ビットバスを持っていますが，グラフィックVRAMをはじめ大部分のI/OデバイスはX68000からの流用を行っているため，16ビット幅に変換された後のバスに接続されています。このため，CPUからこれらのデバイスにアクセスしたときにはX68000と同等のアクセス時間となり，相当数のウェイトがかかることが予測されます。このウェイト数がどの程度なのかを実測してみました。図14に17回連続アクセスしたときの平均値を示します。周辺デバイスに応じて細かくウェイト数を変えており，極力CPUの処理能力の低下を抑えるようにしていることがわかります。

OPMレジスタへの書き込みのウェイト数が極端に多いのは，XVIなどで問題となった，OPMレジスタの書き込み時にBUSYビットをきちんと見ていないアプリケーションに対する措置でしょう。

周辺デバイスへのアクセスタイミングを実測してみると，VRAMなどは比較的アクセスタイミングが安定しているのですが，DMACやSCCといったI/Oデバイスのアクセスは安定しておらず，600nS程度から１μS強までと非常に大きく変動します。

この原因は，I/O関係のバスがCPUクロックではなく，比較的低い周波数のクロックに同期

●図14……各デバイスアクセス時のウェイト数の実測値

デバイス	READ	WRITE
VRAM	9	9
CRTC	9	9
ﾋﾞﾃﾞｵｺﾝﾄﾛｰﾗ	9	11
DMAC	37	31
ｴﾘｱｾｯﾄ	–	7
MFP	24	24
RTC	29	38
プリンタ	7	7
OPM（FM音源）	19	53
ADPCM	18	22
FDC	19	19
SPC（SCSIｺﾝﾄﾛｰﾗ）	18	19
SCC（RS-232Cﾎﾟｰﾄ）	41	49
8255	19	19
ｽﾌﾟﾗｲﾄﾚｼﾞｽﾀ	17	20
ｽﾌﾟﾗｲﾄVRAM	17	18

して動いており，CPUとの同期化のための遅れが生じるためであると考えられます。図に示した値はあくまで参考程度にとどめておくようにしてください。

仮想記憶とMMU

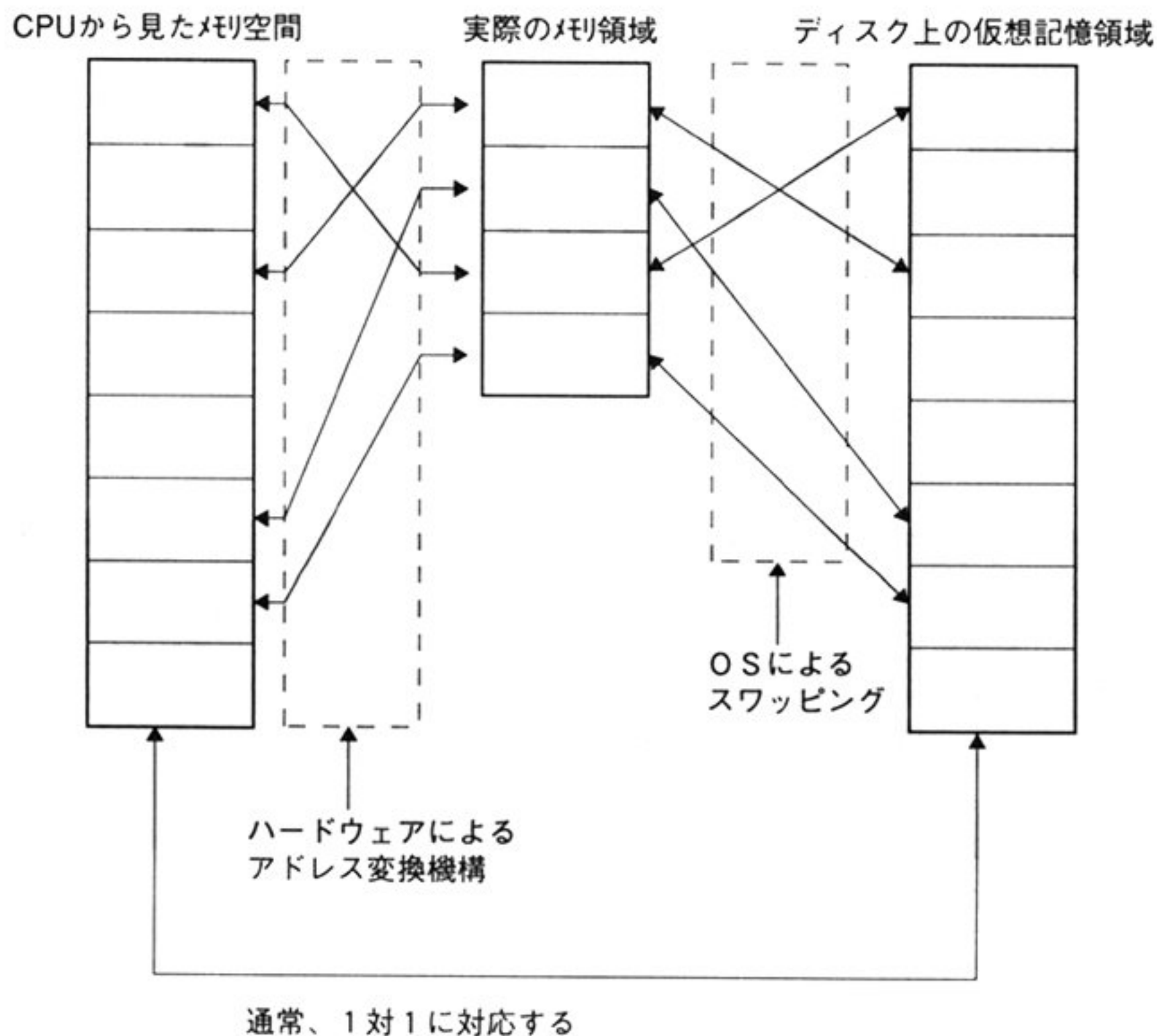
MC68EC030はMMUを使えるようになっていないこともあり，これまではMMUについての説明はしませんでしたが，将来MC68030やMC68040などを使用したシリーズが出ることも考えられるので，ここで仮想記憶の考え方と，M68000ファミリーCPUのMMUについて説明しておくことにします。

1 仮想記憶方式

仮想記憶は，高速ではあるが高価で，容量に制約のあるメモリと，低速ではあるが大容量で低価格なハードディスクなどの補助記憶装置を組み合わせることで得られる，実際には存在しない巨大なメモリ領域です。

CPUは，メモリ上のプログラムを実行するため，通常パーソナルコンピュータではプログラムはすべてメモリ上に配置し，ハードディスクなどの補助記憶装置はファイルを保存しておく場所と考えられています。このため，どれほどハードディスクの容量が大きくても，メモリ容量が少ないと大きなアプリケーションを動かすことはできません。本来は，メモリ容量はできるだけ大きくとりたいところですが，実装スペースなどの物理的な制約やコストの問題などもあって，むやみに大きくすることはできません。

そこで発想を逆転させ，ディスクそのものをメモリとみなし，実装されているメモリはディスク上のコピーであるとして使う方法が考え出されました。つまり，CPUのアクセスできるメモリ空間全体はディスク上にあり，メモリはディスク上にあるメモリ空間の一部を保持し，

●図1……仮想記憶の基本的な考え方

CPUからアクセスできるようにするためのバッファメモリのようなものであると考えるわけです。このディスクとメモリの関係は，ちょうどメインメモリとキャッシュメモリの関係と同じようなものです。

キャッシュメモリのときと同様，CPUがメインメモリに入っていない領域をアクセスした場合には適当なメモリ領域を選択してディスクとの間で入れ替えを行い，その後，CPUにアクセスを再開させるという方法をとることになります。この入れ替え操作を「スワッピング」と呼びます。図1に仮想記憶の基本的な考え方を示します。

1·1 仮想記憶の実現方法

仮想記憶を実現するには，ハードウェアとソフトウェアの両方からのサポートが必須となります。ハードウェア上からのサポートは，アプリケーションが使うアドレス（仮想アドレス）をそのまま実際のメモリのアドレス（物理アドレス）とはせず，特別なレジスタ（このレジスタは通常OSが管理します）などによって物理的なアドレスに変換する機構です。この機能を実現するハードウェアのことをMMU（メモリマネージメントユニット）と呼びます。M68000フ

●図2……仮想記憶動作の例

（1）テーブルの初期値

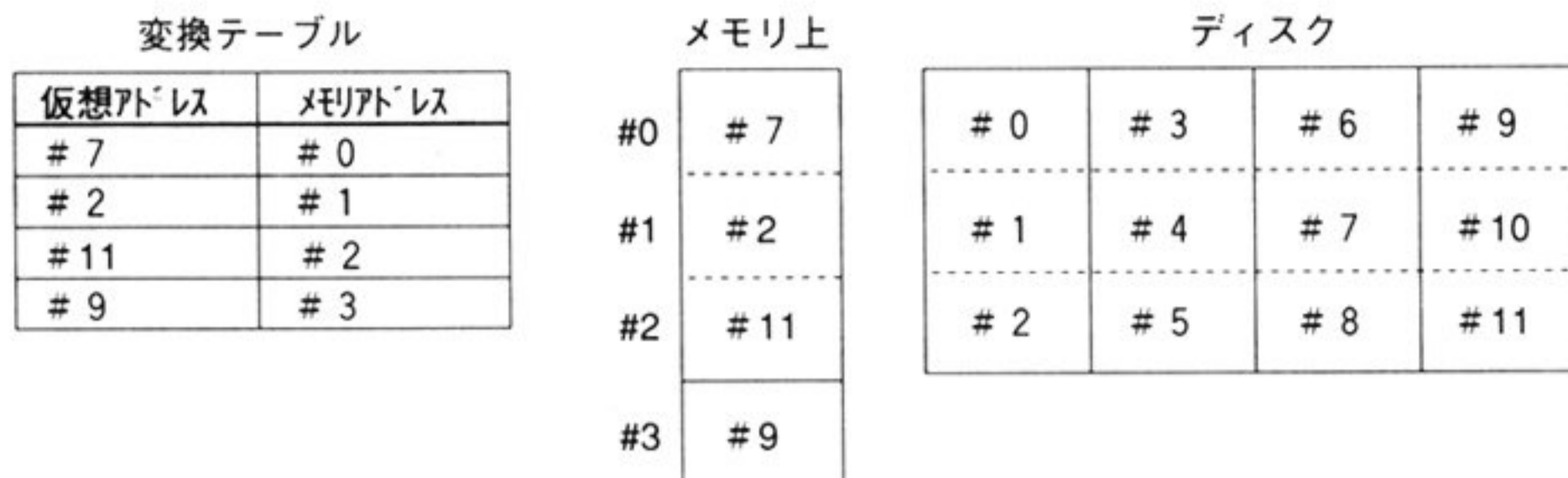
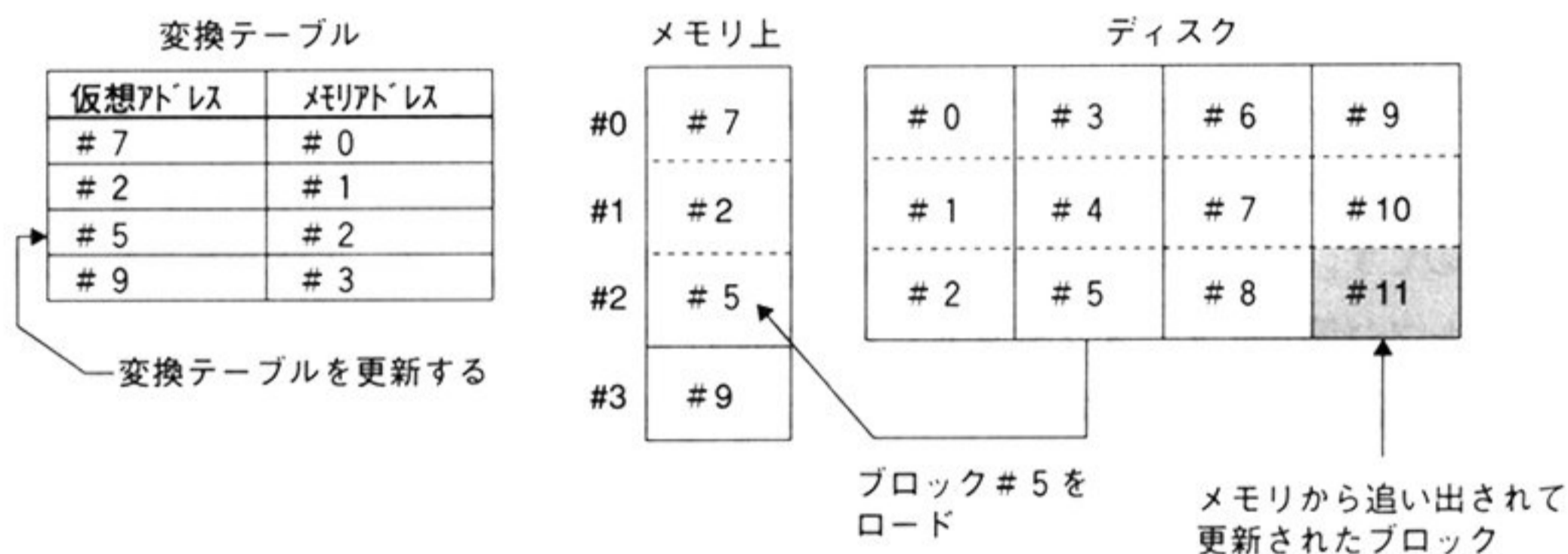
変換テーブル

仮想アドレス	メモリアドレス
＃7	＃0
＃2	＃1
＃11	＃2
＃9	＃3

メモリ上

#0	＃7
#1	＃2
#2	＃11
#3	＃9

ディスク

＃0	＃3	＃6	＃9
＃1	＃4	＃7	＃10
＃2	＃5	＃8	＃11

（2）仮想アドレス＃5をアクセスしようとする。
（3）変換テーブルを探す。
（4）存在しないことがわかる。
（5）適当な領域を選択し（この場合，仮想アドレス＃11：メモリ領域＃2）ディスクに待避する。
（6）追い出してできた空き領域である実メモリ領域＃2に仮想アドレス＃5の内容を読み込む。
（7）変換テーブルを更新する。

ァミリーのMC68020では外付けのMMU（MC68851）が用意され，MC68030以降ではMMU機能がCPUに内蔵されました。

ソフトウェアからのサポートはアクセスされた領域がメモリ上に存在しなかった場合，ディスクなどの補助記憶装置との間で入れ替えを行うことです。通常，これはOSのサービスの1つとして実現されます。

これらの機構の動きはおおむね図2のようになります。まず，アドレス変換機構によって実際のメモリとアプリケーションから見たメモリ番地の関係は切り離され，アプリケーションからアクセスされたアドレスは変換の際に常時チェックされるようにしておきます。

変換機構は，アプリケーションからアクセスされた番地がメモリ上にあるか否かをチェックしており，存在しなければCPUにエラーを通知します（M68000ファミリーでは，この通知にBUSERR信号を使用します）。

エラーの回復処理は通常，OSが行います。OSは変換機構からのエラー通知を知ると，適当な領域をディスクに追い出し，アプリケーションがアクセスしようとした仮想記憶領域の内容をディスクから読み出し，さらに変換機構のレジスタなどの変更を行います。

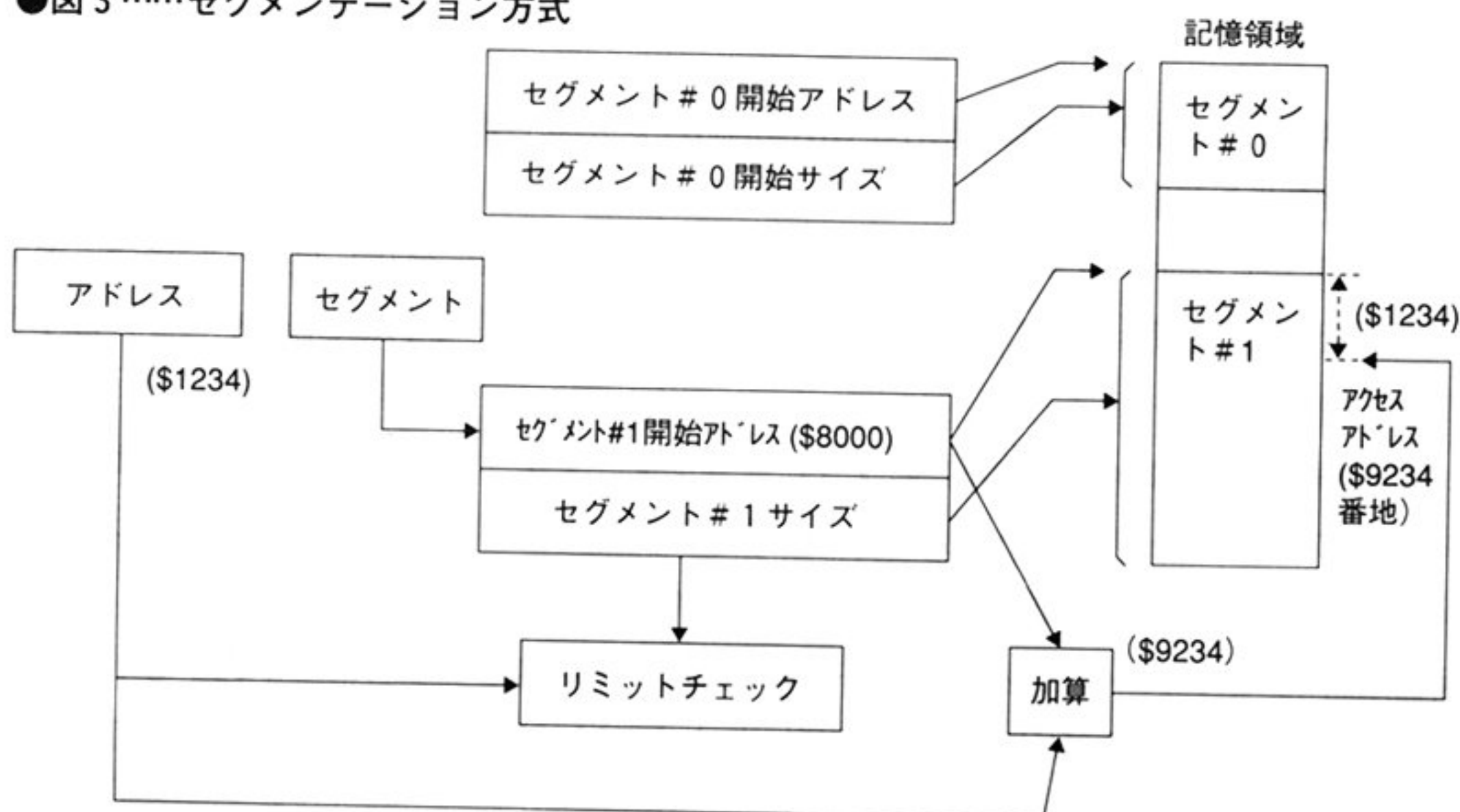
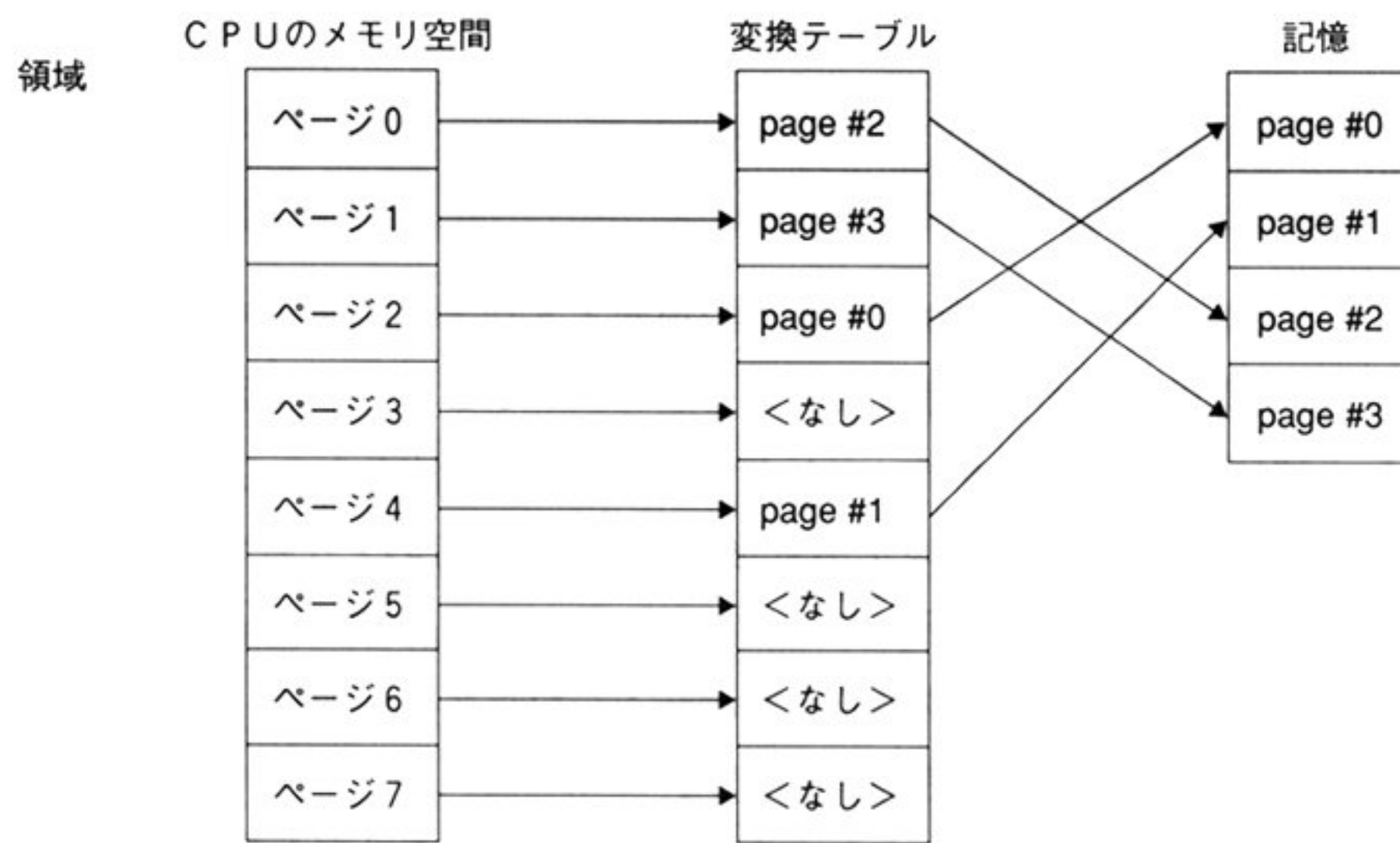
この後，先ほどエラーとなったバスサイクルが再実行されます。このときには新しくメモリ領域が割り振られていますのでエラーは発生せず，アプリケーションの処理が再開されるわけです。

このように，仮想記憶機構はハードウェアとOSの連携があってはじめて可能になるわけです。

1・2 ページングとセグメンテーション

仮想記憶を実現するうえでのメモリ領域の管理方法としてよく使われるのが，ページングとセグメンテーションの2つです。この2つの方式の動作をごく簡単にまとめたのが図3と図4です。

セグメンテーションは，メモリをプログラム領域とデータ領域などといった用途ごとに分類し，それぞれの領域の開始アドレス（ベースアドレス）と確保するメモリ領域の大きさによって管理するものです。OSはベースアドレスと領域長を管理します。一方，アプリケーション側はベースアドレスは関知せず，オフセットアドレスだけを使うようにします。たとえば，ベースアドレスが$8000であるときにアプリケーションが$1234というアドレスを出力したとする

●図3……セグメンテーション方式

●図4……ページング方式

と，アプリケーションからは$1234番地をアクセスしているつもりでも，実際には$9234番地がアクセスされることになるわけです。

各セグメントの内容がメモリ上にあるか否か，どの空間に割り振られているかなどの情報はOSが管理しており，ディスクとメモリの間の入れ替えはセグメント単位で行われます。

セグメンテーションがメモリ領域をその意味ごとに分割しているのに対し，仮想記憶領域を一定の大きさごとに分割しているのが「ページング」です。ページングでは，メモリも「ページ」と呼ばれる単位に分割され，アドレスの変換もページ単位で行われます。あるページがメモリ上に存在するか否かや，実際のメモリアドレスへの対応などはOSが管理します。

セグメンテーションの場合，各セグメントは連続した領域を確保しなければなりませんが，それぞれの大きさはまちまちですので，メモリの確保や解放を繰り返しているとメモリ領域が分断されてしまいます。この状態になると，メモリの総容量は十分あるにもかかわらず，連続した領域として確保できないためにディスクとの間でスワッピングが起こることになります。

ページングの場合は，管理の単位が固定長のページですから，メモリ上では飛び飛びの領域に空きページがあっても，それらを連結して1つのメモリ領域として扱うことができます。このため，セグメンテーションの場合のような無駄は起こりません。

現在，コンピュータの仮想記憶方式としてはページングが主流であり，セグメンテーションを行っているものはあまりありません。M68000ファミリーのCPUの仮想記憶機構もページング方式です。また，他のマイクロプロセッサでもRISC，CISCなどの種別を問わず，ほとんどのものがページング方式をサポートしています。インテルの80286は例外的にセグメンテーションによる仮想記憶をサポートしたCPUでしたが，これも80386以降ではページング機構が導

入されています。

1・3 ページの入れ替え

通常，スワッピングはCPUがメモリ上に置かれていない仮想アドレスをアクセスした場合に発生します。このように，実際のアクセス要求があったときにスワッピングを行う方法を「デマンドページング方式」と呼びます。デマンドページング方式は最も一般的なもので，MC68020以降のCPUの仮想記憶機構もデマンドページング方式をサポートするものとなっています。

マルチタスク機構を実現したシステムの場合，タスクごとに使用する記憶領域がまったく異なってきます。このため，仮想記憶とマルチタスクをサポートしたシステムでは，タスクが切り替わった後にスワッピングが頻発する可能性があります。

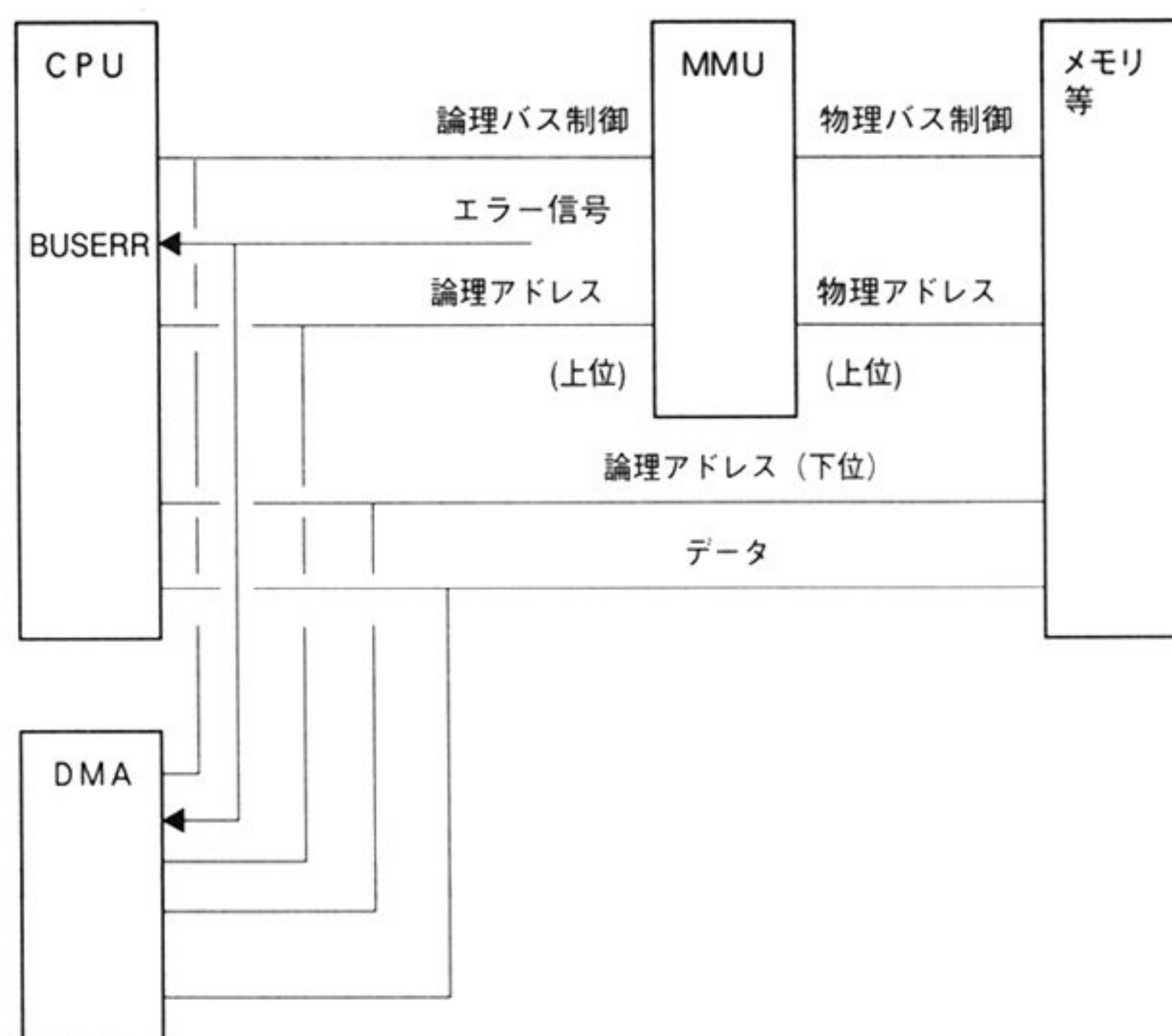
この影響による性能低下を抑えるため，必要となりそうなページをあらかじめ入れ換えてしまうという方法が考えられました。この方法のことを「プリ・ページング」や「ルックアヘッドスワッピング」などと呼ばれています。

ただ，実際に必要となるメモリ領域を予測するのはかなり困難ですので，実際のシステムに導入されているのは，ほとんどがデマンドページング方式です。

2 MMU

仮想記憶を実現するための，アドレス変換やアクセス監視を行う部分を「MMU」（メモリマネージメントユニット）と呼びます。MMUを使ったシステムの概略を図５に示します。MMUを使用したシステムでは，メモリなどに与えられるアドレス値はCPUが出力したアドレスではなく，CPUから受け取ったアドレスをMMUが変換したものになります。このことからCPUが出力したアドレスを「論理アドレス」（または仮想アドレス），MMUから出力されるアドレスを「物理アドレス」と呼んで区別しています。

通常，MMUはCPUのアドレスを変換するために，変換用のテーブル（ページテーブル）を参照します。この変換テーブルには論理アドレスと物理アドレスの対応のほか，その領域をアクセスしてもよいか，書き込みを行ってもよいかなどの情報が含まれています。MMUは，アドレス変換時にこの情報を参照し，もしそのまま物理アドレスを出力できない場合にはCPUに対してエラーを通知し，メモリへのアクセス動作を行わせないようにします。

●図5……CPUとMMU、メモリの接続関係

CPUは，MMUからのエラーの通知を受けると例外処理に移行します。ソフトウェア(OS)は例外処理のなかで変換テーブルを書き直してエラーになったアクセスを再開させたり，例外を起こしたプログラムを強制的に終了させるなどの回復処置をとるわけです。

MC68000では，バスエラーを受けたときにスタックに退避されるCPUの内部情報が不十分なため，エラーとなったバスサイクルを再実行することができません。このため，MC68000にMC68851をつけても仮想記憶は実現できません。MC68010では，この問題が修正され，バスサイクルの再実行が可能となっています。

3 MC68851の仮想記憶機構

M68000ファミリーCPUのMMU機能はMC68020の外付けMMUとして開発されたMC68851でほぼ完成されており，MC68030やMC68040で内蔵されたMMUはMC68851の機能を簡略化したものとなっています。ここでは，MC68851の機能について説明した後，MC68030，MC68040の内蔵MMUについて説明します。

3·1 MC68851の機能

MC68851は，ページング方式による仮想記憶機構を実現するコプロセッサです。MC68851の持つ機能を大きく分けると，次のようになります。

1)ファンクションコードによるアクセス領域の区分け

2)スーパーバイザモード／ユーザモード/DMAによるアクセス領域の区分け

3)ページテーブル検索

4)アドレス変換

5)アクセス上下限チェック

6)書き込み禁止（保護）

7)アクセスレベル管理

8)ブレークポイント機構

このうち，8)のブレークポイント機構はデバッグ用に設けられたものですので，仮想記憶動作に直接関係するのは1)～7)ということになります。

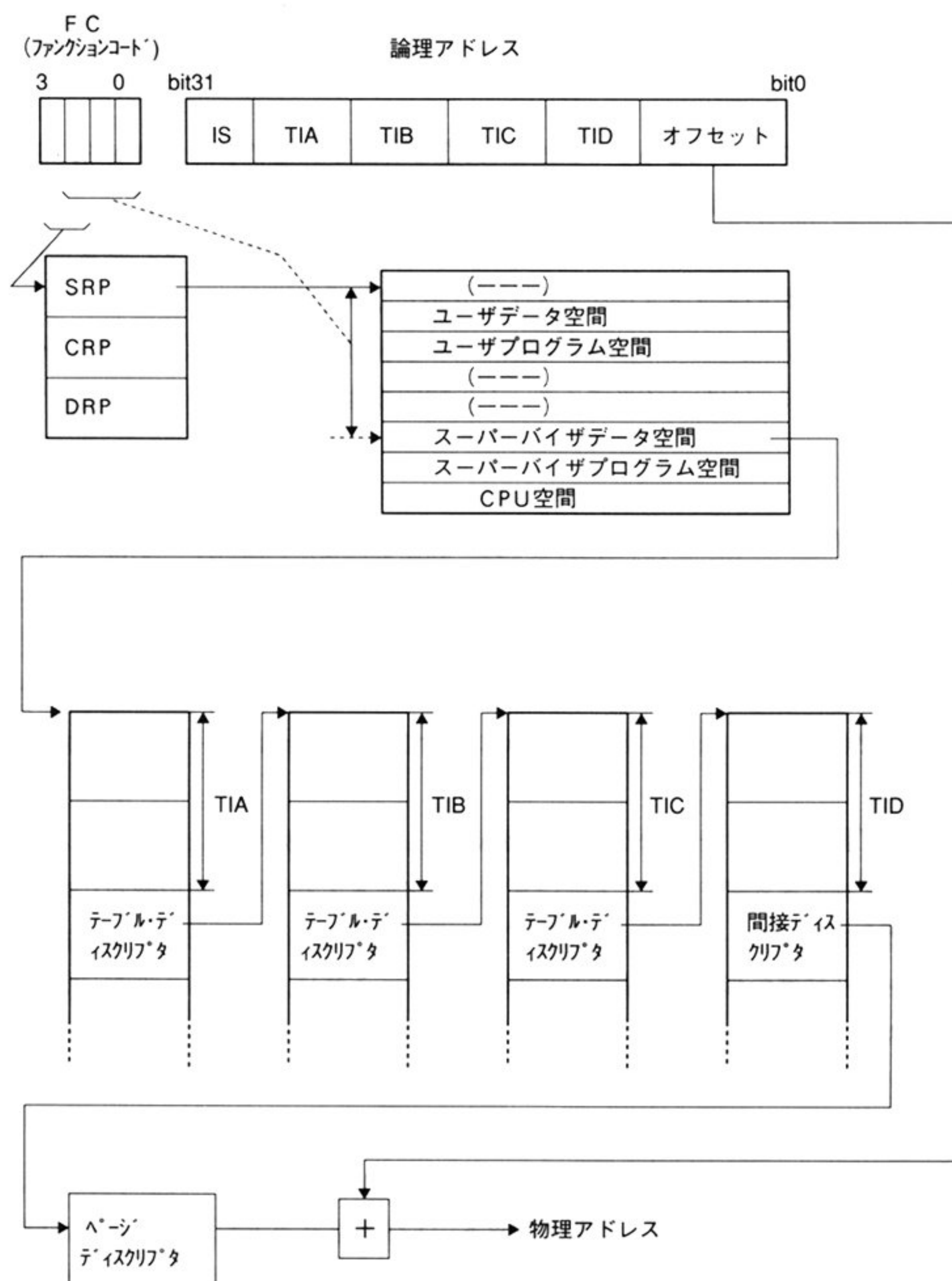
3·2 MC68851のアドレス変換動作の概略

ページング方式を実現するうえで鍵となるのは，与えられた論理アドレスやファンクションコードなどの情報から，それに対応する物理アドレスを記述してあるテーブル（ページテーブル）を検索するかどうかということです。

MC68851のページングの方法を図示したのが図6です。ページディスクリプタ(ページテーブル）の検索に用いるのは32ビットのアドレス線と，4本のファンクションコード信号です。ファンクションコードのうち下位3ビットはCPUが出力するもので，アクセス動作がスーパーバイザなのかユーザなのか，データなのかコードなのかといった情報が与えられます。ファンクションコードの最上位ビット（FC3）は，アクセスがCPUによるものなのかDMAによるものなのかを示すものです。

MC68851がこれらの情報からページテーブルを検索する基本的な方法は次のようになっています。

①まず，与えられたファンクションコードの上位2ビットによって，ページ検索を開始する番地の選択を行います。このため，MC68851は3本のレジスタ（ルートポインタ）を持っています。SRP(スーパーバイザルートポインタ）はスーパーバイザ用，CRP(CPUルー

●図6……MC68851のページング機構

トポインタ）はユーザ用，DRP（DMAルートポインタ）はDMA用です。

②次に選択されたルートポインタで指す番地にファンクションコードの下位3ビットで決まるオフセットを加算します。これにより，ファンクションコードごとに仮想空間を分割することができるわけです。

③一方，32ビットあるアドレス信号のうち，ISで示されるビットは無視され，残るビットが

4つのテーブルインデックスと，1つのオフセットに分割されます。ISで上位ビットを無視できるようにしているのは，MC68851がアドレスバスを32本未満のCPUと接続されることも考慮しているためです。MC68020はアドレスバスを32本持っていますので，通常ISビットは使用しません。

④②で得たベースアドレスから始まるテーブルから，③で分割されたアドレスのうち，TIA番目のエントリの中身を引き出します。

⑤次に④で得られたアドレスをベースアドレスとして，TIB番目のエントリを，さらにその中身が指す先のTIC番目のエントリ……と順次取り出していきます。

⑥最終段である，TIDで指定されるエントリが指している先はページディスクリプタ（ページテーブル）であり，このなかに該当するページの開始番地（物理アドレス）が書いてあります。

このアドレスと論理アドレスのオフセット領域の値が加算されてアクセスすべき物理アドレスが得られます。

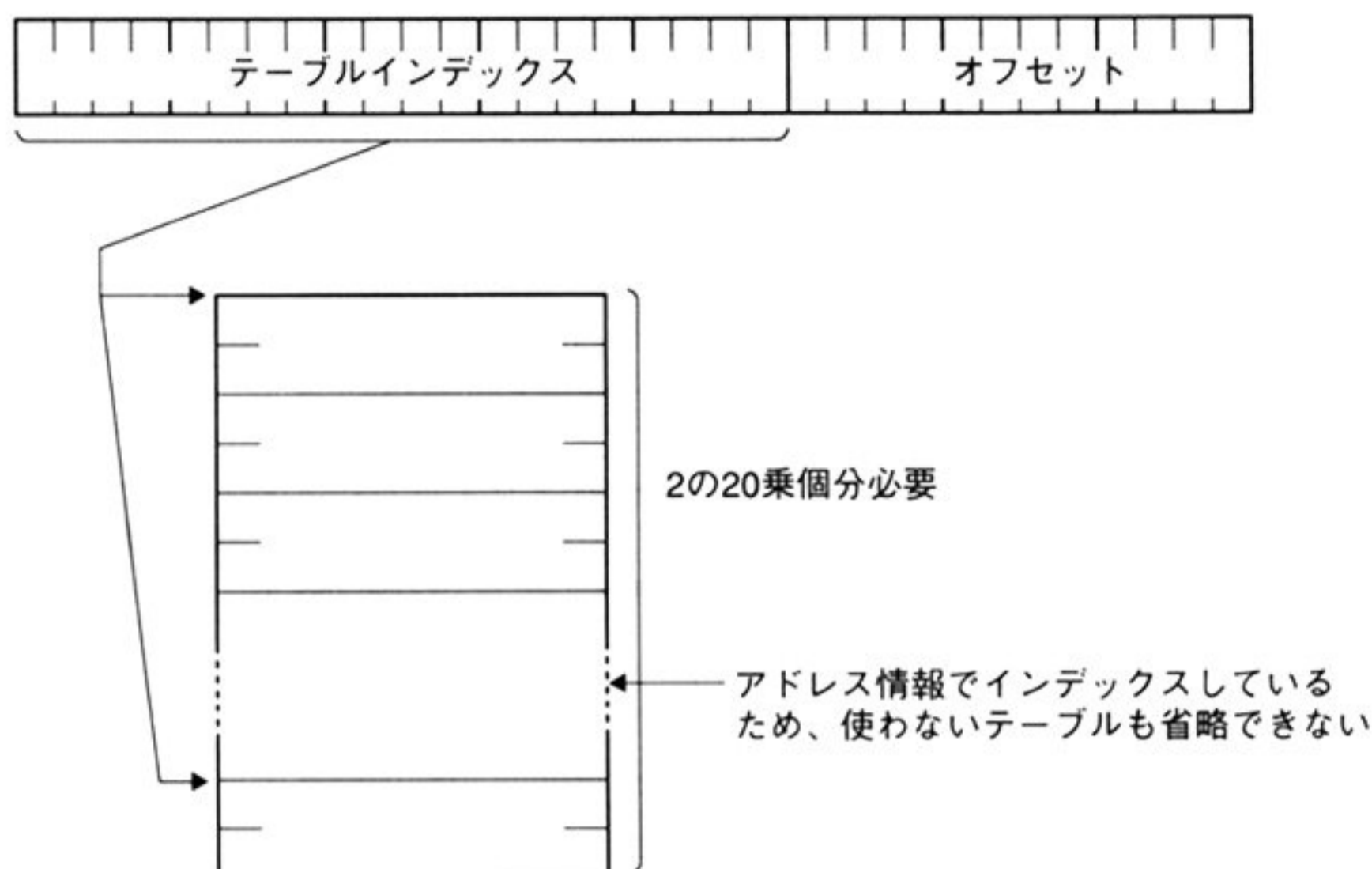
ファンクションコードによるルートポインタレジスタの使い分けやテーブル検索などを行うか否か，ISやTIA，TIBなどのビット数をいくつにするかといったことはすべてプログラマブルです。ITB～TIDを0としてしまうことももちろん可能で，この場合はテーブル検索の段数が少なくなります。

④以降のステップで使われるテーブルのうち，最終段のものを「間接ディスクリプタ」，それ以前のものを「テーブルディスクリプタ」と呼びます。アクセスするエントリが無効であるか，ページディスクリプタであるか，テーブル／間接ディスクリプタであるかなどといった情報は，テーブル内の各エントリに書かれている情報から得られます。これにより，1つのテーブルのなかにページディスクリプタとテーブルディスクリプタ（または間接ディスクリプタ），無効ディスクリプタなどが混在することが許されます。

もしテーブルを検索している途中でページディスクリプタに出会うと，MC68851はそのディスクリプタに書かれているベースアドレスと，残っている下位ビットで示されるアドレスを加算して物理アドレスとします。たとえば，ISが0ビット，TIAが8ビットであったとき，TIAを使って引いたテーブルにページディスクリプタがあれば，ページディスクリプタに書かれているアドレスに論理アドレスの下位24ビットの値を加算して物理アドレスとします。

また，途中で無効ディスクリプタに出会った場合はMC68851はテーブル検索を中断し，CPUに対してテーブルが存在しないことを通知します。

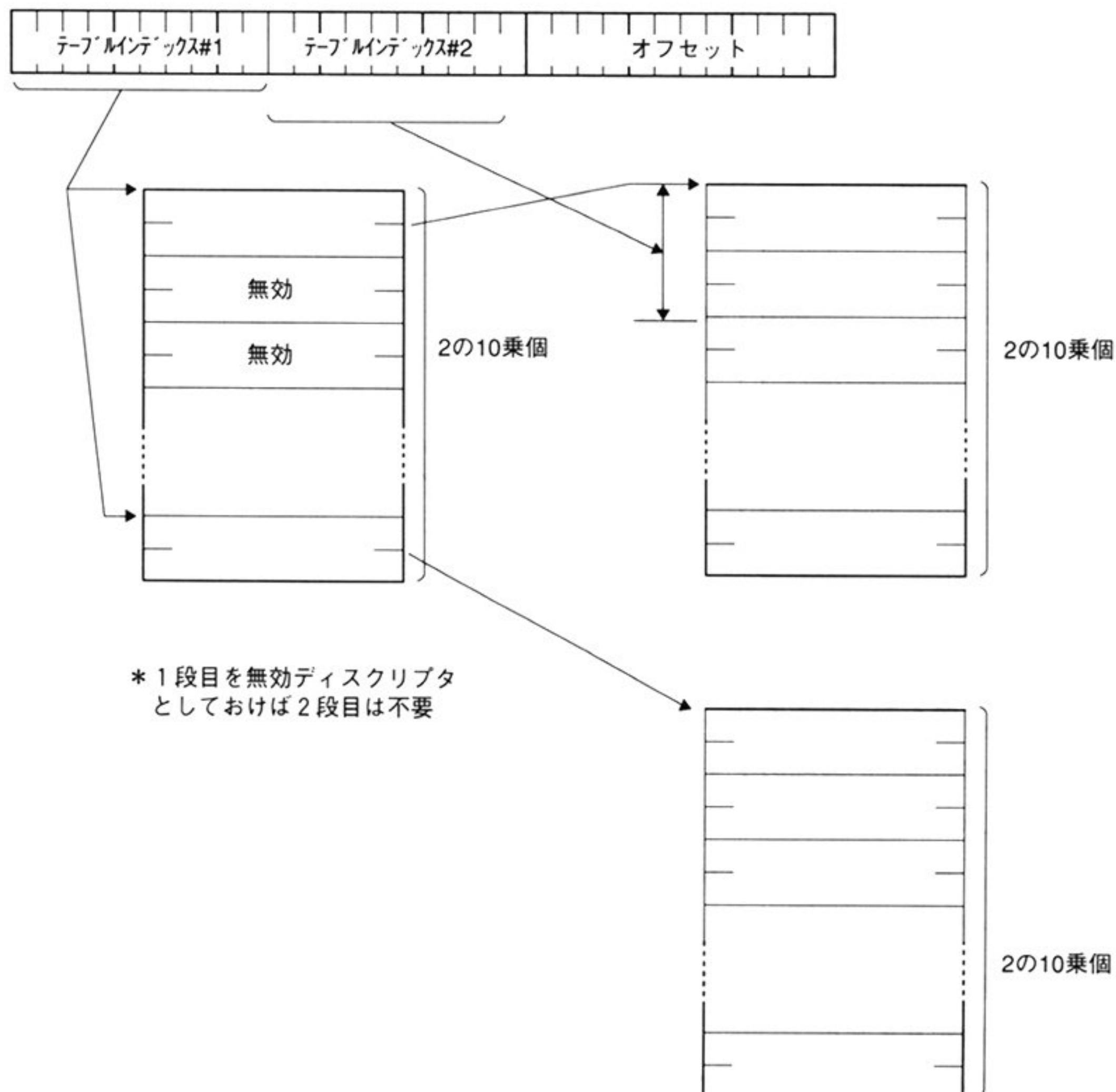
MC68851がテーブルディスクリプタを何段も使ってアクセスするようになっているのは一見複雑で面倒なようですが，これはテーブルのサイズを圧縮するためにとられた方法なのです。論理アドレスが32ビット，ページサイズ（1ページあたりのバイト数）を4K（アドレス12ビッ

●図7……アドレスから直接テーブルを引く場合

ト分）とすると，残る20ビットを使ってテーブルを検索することになります。

もしページングの原理の説明のように，この20ビットでいきなりテーブルを引くようにしたとしましょう。このとき，プログラムが0番地付近と$FFFFFFFF番地付近の2ページしか使わないとしても，テーブルは全アドレス分，すなわち2の20乗分が必要になります（図7参照)。もし，テーブルの各エントリが1つあたり4バイトで作られていると，テーブル全体のサイズは2の24乗バイト，すなわち16Mバイトが必要になってしまいます。これでは何のための仮想記憶なのかわかりません。

ここで，20ビットのアドレスを10ビットずつに分割してみます(図8参照)。このとき，1段目のテーブルは2の10乗（＝1024）個ですから4Kバイトとなります。アクセスされる領域が0番地付近と$FFFFFFFF番地付近しか使わないのであれば，このテーブルの先頭と最終以外のところはすべて無効ディスクリプタにしてしまい，2段目のテーブルを用意する必要はありません。

2段目のテーブルもそれぞれ4Kバイト必要です。この例の場合，2段目のテーブルは2つしか必要ありません。つまり，1段目と2段目のテーブルをあわせても12Kバイト分のテーブルがあればよいことになるわけです。

●図８……アドレスを２分割してテーブル量を節約

3・3 アドレスの上下限チェック

テーブルディスクリプタは，次段のテーブルの先頭アドレスに加えて，次段のテーブルを引く際のインデックスの上限値，または下限値を設定することができるようになっています。

また，本来ならテーブルディスクリプタがあるテーブル内にページディスクリプタを置いた場合にも，そのページをアクセスする際のオフセットの上限値，または下限値を設定することができるようになっています。

3・4 書き込み禁止（保護）

命令や重要な定数データが置かれているなどの理由で，あるページに対する書き込み動作を禁止したい場合のために，テーブルディスクリプタやページディスクリプタには書き込み禁止フラグが設けられています。

書き込み動作を行うとき，MC68851は書き込み禁止とされているディスクリプタに出会うと，それ以降のテーブル検索を中断し，エラーの発生をCPUに通知します。

テーブルディスクリプタにも書き込み禁止を設定できるため，広い領域をまとめて書き込み禁止にすることも可能です。

3・5 アクセスレベル管理

M68000ファミリーCPUはユーザモードと，スーパーバイザモードという2レベルの動作モードを持っており，ユーザモードではシステムに影響を与えるような命令の使用が制限されるなどの保護機構が働くようになっています。このようなレベル管理をさらにきめ細かく行えるように考えられたのが，MC68851で導入された「アクセスレベル」です。

MC68851は，各ページディスクリプタやテーブルディスクリプタ，アクセスする論理アドレス空間，そして現在動いているプログラムのそれぞれについて最大8段階の特権レベルをつけることができるようになっています。レベル0が最も高いレベル，レベル7が最も低いレベルになります。

特権レベルの低い状態では，高い特権レベルを設定された領域はアクセスできません。

3・5・1 アクセスレベル保護の概要

MC68851によるアクセスレベル保護は次のような3段階のステップで行われます。

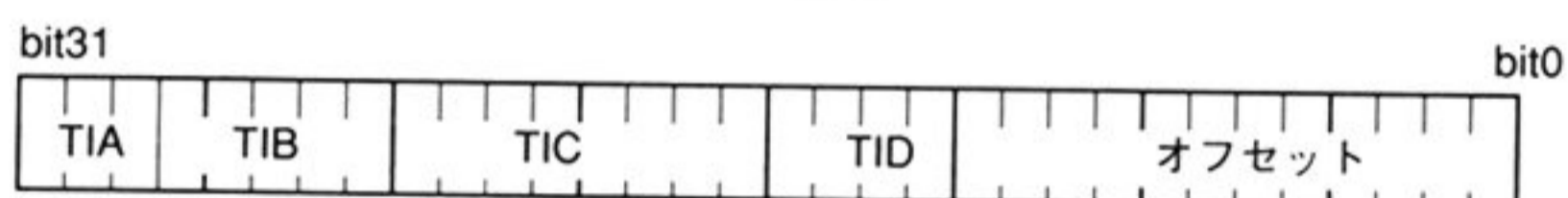
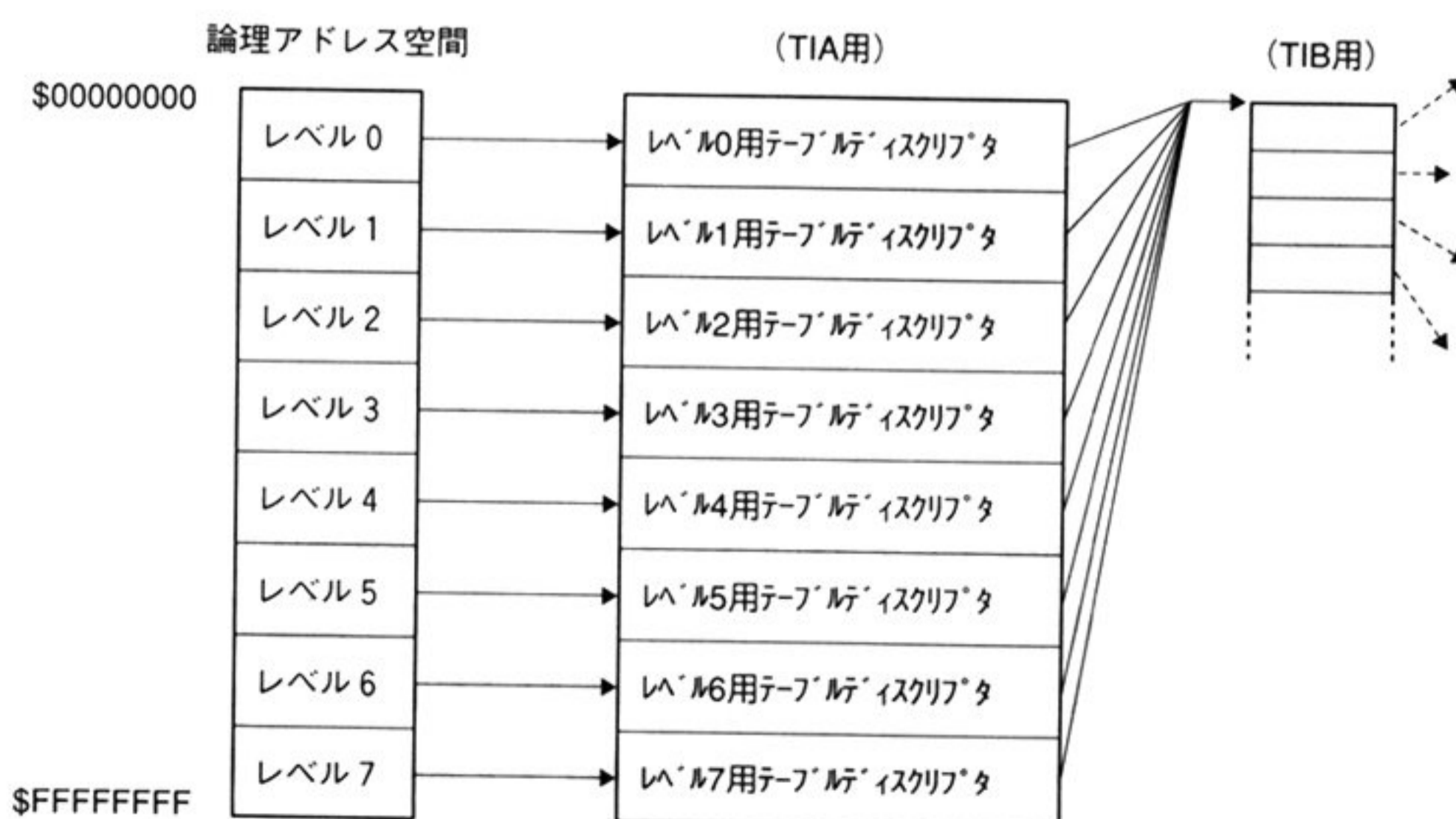
①まず，ファンクションコードがチェックされます。アクセスレベル保護は，ユーザモードでのアクセスに対してのみ働き，スーパーバイザモードやDMAでの転送では働きません。
②次に，現在の動作レベルを示すMC68851のレジスタ（CAL：Current Access Level レジスタ）と，論理アドレスのレベルを比較します。論理アドレスのレベルは，論理アドレスの上位ビットを取り出したものです。

③これ以降，テーブル検索に入りますが，このとき，論理アドレスで示されるレベルと各ディスクリプタ（ページディスクリプタやテーブルディスクリプタ）に設定されているレベルが比較されます。

論理アドレスの上位ビットはテーブル検索のためのインデックスとして使われるにもかかわらず，レベルとしても使用されることが不思議に思われるかもしれませんので少し補足しておきましょう。

論理アドレスの上位3ビットがアクセスレベルになるということは，論理アドレス空間の全体を8等分し，0番地側から順にレベル0用の領域，レベル1用の領域，レベル2用の領域……と分離していることにほかなりません。

論理アドレスをこのように分割していても，MC68851はページング機構を使っていますから，異なるアクセスレベルの領域に対して同一の物理アドレスをマッピングすることも可能です。たとえば，TIAのビット幅を3として，そのなかのテーブルディスクリプタに書かれている次段のテーブルの先頭アドレスをすべて同一にしてしまったとしましょう。この場合，アドレスの上位3ビットは事実上論理アドレスとしての意味がなくなり，レベル管理用のビットとしてのみ扱われることになります。この状態を図9に示します。

●図9……アクセスレベルビットの利用

このようにした場合，どの論理アドレスからでも同じ領域が見えるわけですから，各ページに対するアクセスレベル保護が必要となります。これが先ほど示したレベル保護のステップの③になります。各ディスクリプタに対して，どのレベルからのアクセスを許すかを設定できるようにすることで，きめ細かなレベル管理ができるようになるわけです。

3・5・2 レベル間のアクセス保護

MC68851のレベル間のアクセス動作に対する保護は次のようになっています。

1) ある特権レベルにあるプログラムは，自分と同じレベル以下のレベルのページにしかアクセスできません。
2) 通常のBSR，JSRなどの命令で呼び出せるのは，自分と同じレベルのプログラムだけです。
3) 自分より低いレベルのプログラムの呼び出しはできません。
4) 自分より高いレベルのプログラムの呼び出しを行うためにはCALLM（CALL Module）という専用命令を使用します。CALLMで使うことができるのはページディスクリプタで，かつプログラムのエントリポイントを示すテーブル（「モジュールディスクリプタ」と呼びます）が置かれていることが示されている（Gビットが‘1’になっている）ディスクリプタだけです。
5) 自分より高いレベルのプログラムへの復帰（RTSなど）はできません。
6) 自分より低いレベルのプログラムへの復帰には，RTM（ReTurn from Module）命令を使用します。

レベルの上下関係のイメージとしては，OSとアプリケーションの関係を考えるとわかりやすいでしょう。OSにはアプリケーションよりも高いレベルを与えます。OSの命令領域やデータ領域をアプリケーションが破壊したり，無条件に参照できるようでは問題がありますから，レベルの高いOS領域はレベルの低いアプリケーションからはアクセスさせないようにする必要があります。また，OSはアプリケーションが用意したバッファにデータを書き込むなどのサービスが必要ですから，OSからはアプリケーション領域へのアクセスができなければなりません。これが先ほど掲げた1）に相当します。

また，OSがアプリケーション側で用意したサブルーチンを利用するような切り口を与えてしまってはOSの信頼性に問題が発生します。一方，アプリケーションはOSのサービスプログラムを呼び出す必要がありますが，この際，無制限にOSの内部処理ルーチンなどに飛び込まれる

ようではやはりOSの信頼性に問題が発生します。このため，OSのサービスルーチンはあらかじめ決められた入り口（ゲート）以外からは呼び出せないようにしなくてはなりません。これが2)，3)，4)に相当します。

呼び出しが同一以上のレベルに対するものしかない以上，復帰は自分以下のレベルのものに対するものしかないはずです。また，もし自分よりも高いレベルに対する復帰が許されるようだと，スタックに適当な内容をセットしてRTM命令などを実行することで簡単にOSのなかの任意の場所に飛び込むことが可能となってしまい，問題があります。これらに対処したのが5)と6)というわけです。

3・5・3 リードアクセスレベル／ライトアクセスレベル

各ディスクリプタのレベルは，リード時とライト時で別々のものを設定でき，それぞれ「リードアクセスレベル」と「ライトアクセスレベル」と呼んでいます。リードアクセスでは，現在の動作レベルとリードアクセスレベルが比較され，ライト時にはリードアクセスレベルとライトアクセスレベルのうち，より上位のレベルの側と比較されます。

これにより，たとえば，書き込みは自分と同等以上のレベルのものしか行えないが，内容の読み出しは下位のものにも解放するという使い方ができるようになります。

ライトアクセスレベルをそのまま解釈してしまうと，ライトアクセスレベルをリードアクセスレベルよりも低く設定すると，低いレベルのものからは書き込めるが，読み出しはできないという妙なことになってしまいます。このため，書き込み時にはリードアクセスレベルと，ライトアクセスレベルのうち，高い側を採用するようにしているわけです。

3・6 ブレークポイント機能

M68000ファミリーCPUには8個のブレークポイント命令($4848～$484F)が用意されています。CPUはこれらの命令に出会うと，ブレークポイント番号（0～7）をアドレスバス上に示し，ブレークポイントアクノリッジサイクルを実行します。

このとき，周辺デバイスが応答すれば，CPUはそのときデータバス上にあるデータを命令(「置換オペコード」と呼びます)として実行します。また，バスエラーになると無効命令例外(例外番号$4）として例外処理に移ります。

MC68851は，CPUの持つブレークポイント命令をサポートするため，8つのブレークポイント番号それぞれに対応して，ブレークポイントのスキップカウンタと，置換オペコードを保持

するためのレジスタを持っています。

ブレークポイントを利用するときには，仕掛けたい場所の命令を置換オペコード用のレジスタにコピーした後，メモリ上の命令をブレークポイント命令に書き換え，さらにその場所を何回通過したらブレークさせるかをスキップカウンタに書き込んでおきます。

CPUがブレークポイント命令を実行するたびに，MC68851のスキップカウンタが1ずつ減らされていきます。カウント値が0でなければ置換オペコードがCPUに送られますので，CPUは何事もなかったようにプログラム実行を継続します。

スキップカウンタが0になると，MC68851は置換オペコードの送出は行わず，バスエラーでCPUに応答します。この結果，CPUは無効命令例外処理に移ります。

無効命令例外処理のなかで要因がブレークポイント命令によるものであることがわかれば，その時点でデバッガに移行するなり，該当タスクの動作を止めるなりの処理を行えばよいわけです。

4 MC68851の内部レジスタ

4・1 MC68851のレジスタの概要

MC68851のうち外部からアクセス可能なレジスタの一覧を図10に示します。また，それぞれのレジスタの機能の概要は次のようになっています。

1)ルートポインタレジスタ（64ビット長）

変換テーブルの先頭アドレスなどを保持するレジスタで，次のものがあります。

・CRP（CPUルートポインタ）

・DRP（DMAルートポインタ）

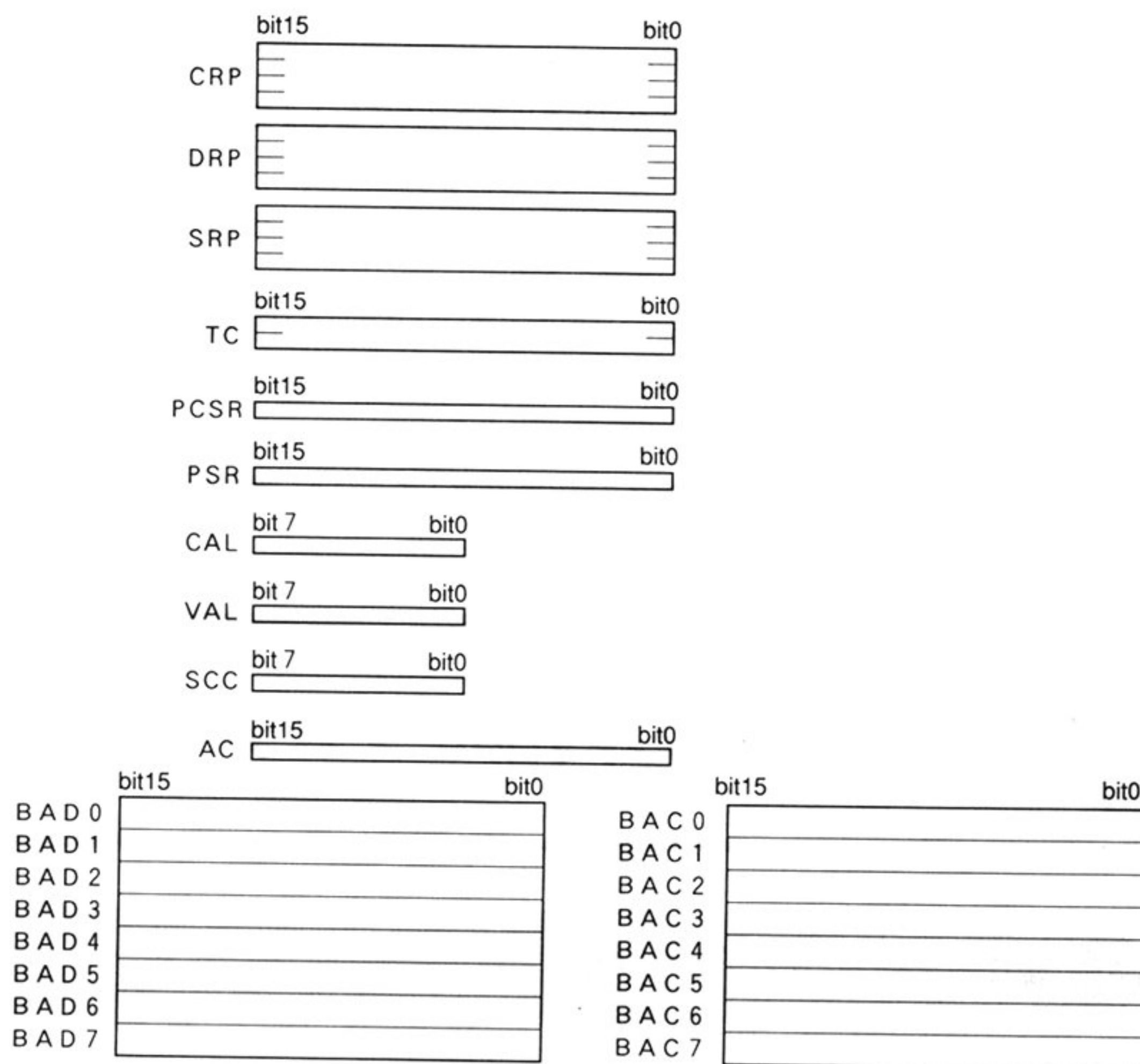
・SRP（スーパーバイザルートポインタ）

2)変換制御レジスタ（TC：16ビット長）

ISやTIA，TIBなどのビット長，ページの大きさ，ファンクションコードを使うか否かなどの，テーブル検索の方法を指定するレジスタです。

3)キャッシュステータスレジスタ（PCSR：16ビット長）

ページング機構は変換テーブルに基づいて動作しますが，メモリアクセスのたびにテーブ

●図10……MC68851のレジスタ

ルを読み出していたのでは効率が著しく低下するため，MC68851は64エントリ分の変換テーブル用のキャッシュメモリ（ATC：アドレス変換キャッシュ）を持っています。このキャッシュメモリの状態などを示すのがPCSRです。

4）ステータスレジスタ（PSR：16ビット長）

指定した論理アドレスのステータスやアクセス権の情報が反映されるレジスタです。

5）アクセスレベル保護用レジスタ（8ビット長）

アクセスレベルの保持に使われるもので，次のものがあります。

・CAL（カレントアクセスレベル）

・VAL（有効アクセスレベル）

・SCC（スタック変更制御）

CALは現在のアクセスレベルを保持するもの，VALはCALLM命令で呼び出しが行われた場合，呼び出した側のレベルを保持するレジスタです（呼ばれた側はCALに入りま

す)。SCCは，CALLM命令などでレベル間の移動があったとき，スタックを変更する必要があるか否かを指示するものです。

6)アクセス制御レジスタ（AC：16ビット長）

アクセスレベル保護機能を使用するか否か，使用するときには何レベルまで使用するかなどを指定するためのレジスタです。

7)ブレークポイントアクノリッジデータレジスタ（BAD0～BAD7：16ビット）

ブレークポイント動作を行うとき，置換オペコードを保持するために使用されます。

8)ブレークポイントアクノリッジ制御レジスタ（BAC0～BAC7：16ビット）

該当するブレークポイント命令に対し，MC68851がブレークポイント動作を行うか否かや，何回目のブレークポイント命令でバスエラーを発生させるかなどの指定を行います。

4・2 ルートポインタレジスタ

CRP，SRP，DRPの各ルートポインタレジスタのビット配置を図11に示します。

●図11……ルートポインタレジスタのビット配置

・L/U　1：リミットフィールドの値は下限値
　　　0：　〃　上限値
・SG　1：グローバル共有する
　　　0：　〃　しない
・DT　00：無効
　　　01：ルートポインタの指す先に変換テーブルはない
　　　10：ルートポインタの指す先にショートフォーマットの変換テーブルがある
　　　11：ルートポインタの指す先にロングフォーマットの変換テーブルがある

・リミット　テーブルインデックス値の制限値
・ﾃｰﾌﾞﾙｱﾄﾞﾚｽ　DT=01のときは論理アドレスに対するオフセット値
　　　DT=10または11のときは次段のテーブルの先頭アドレス

4・2・1 L/U（下限／上限）ビット

リミットフィールドの値を，テーブルをインデックスする際の上限値とするか下限値とするかの指定を行います。L/Uビットが'1'のときにはリミットフィールドの値はインデックスの下限値と解釈されます。インデックスの値はリミットフィールドの値以上でなければなりません。

L/Uビットが'0'のときには逆に，インデックスの値はリミットフィールドの値以下でなければなりません。

TCレジスタのFCL(ファンクションコードルックアップ)ビットが'1'になっているとき(イネーブルされているとき）には，このビットは意味を持ちません。

4・2・2 リミットフィールド

テーブルサーチで使用するオフセット値の上限値／下限値を設定するために使用します。

リミットチェックを無効にしたいときにはL/Uビットを'0'(上限値）としてリミットフィールドを最大（$7FFF）にするか，またはL/Uビットを'1'（下限値）としてリミットフィールドを最小（$0000）にします。

4・2・3 SG（グローバル共有）ビット

マルチタスクを行っているシステムでは，タスクごとに異なる論理空間を持たせるのが普通です。このため，CRPはタスク切り替えのときに変更されることが少なからずあります。一方，MC68851はアドレス変換を高速化するため，論理アドレスから物理アドレスへの変換結果を保持するATC（アドレス変換キャッシュ）を持っています。ATCがCRPの変更を知らないと，過去のCRPに対する変換テーブルの情報に基づいて変換が行われてしまいます。

これを避けるため，CRPが変更されるたびにATCをクリアしてしまうのは簡単ではありますが，あまりに無駄が多い方法です。このため，MC68851には最近参照されたCRPの値を8個まで保存しておくRPT（ルートポインタテーブル）があり，それぞれに対応した番号（TA：タスクエイリアス）がつけられています。ATCのほうもそれぞれにTAがつけてあり，現在のタスクエイリアス値と一致するものだけが有効とみなされます(RPTやタスクエイリアスの管理はMC68851が自動的に行うので，プログラマはほとんど気にする必要はありません)。

このようにTAによる管理が行われると，今度はタスク間で同じ領域を共有する場合に，TA

が違うだけでまったく同じ変換テーブルがATCにいくつも保持されることが起こります。これではATCの使用効率が悪くなります。DRPやSRPによって生成されたATCのエントリにもTAの現在値がセットされますので，同じものが作られてしまう可能性はさらに高くなります。

これに対処すべく，異なるTAを持つものであっても同じ変換テーブルを共有できるようにする手だてとして用意されたのがSGビットです。SGビットはページディスクリプタやテーブルディスクリプタにも存在します。ATCにSGビットが立ったエントリが作成された場合，TAの一致判定が省略され，タスク間で1つのエントリを共有することができるようになります。

4・2・4 DT（ディスクリプタタイプ）フィールド

DTフィールドは，そのルートポインタレジスタのテーブルアドレスフィールドの指す先にあるものがショートフォーマットのディスクリプタであるか，ロングフォーマットのディスクリプタであるか，あるいはルートポインタレジスタそのものがページディスクリプタであるかを示します。

DTが‘00’のときは無効扱いとなっていますが，ルートポインタが無効というのは許されませんので，ソフトウェアから‘00’に設定することはできないようになっています。

DTが‘01’のときは，ルートポインタレジスタに入っている内容がいきなりページディスクリプタであるということになります。このとき，与えられた論理アドレスにテーブルアドレスフィールドの値を加えたものが物理アドレスとなります。

4・2・5 テーブルアドレス

DTフィールドが‘01’のときは，物理アドレスのオフセット値を示します。この場合，論理アドレスにテーブルアドレスの値が加算されて物理アドレスとなります。

DTフィールドが‘10’，または‘11’のときには次段の変換テーブルの先頭アドレス（物理アドレス）を示します。

4・3 TC（変換制御）レジスタ

TCレジスタは，MMUのアドレス変換機構のイネーブル／ディセーブルや，論理アドレスの分割方法の指定などを行います。

●図12……TC（変換制御レジスタ）のビット配置

E	0	SRE	FCL	PS	IS
TIA	TIB			TIC	TID

- E　0：変換機構ディセーブル
  1：変換機構イネーブル
- SRE　0：スーパーバイザルートポインタを使用しない
  1：　〃　使用する
- FCL　0：ファンクションコードをテーブルインデックスとして使用する
  1：　〃　使用しない
- PS　$0～$7：未使用
  $8：ページサイズは256バイト
  $9：ページサイズは512バイト
  $A：ページサイズは1Kバイト
  $B：ページサイズは2Kバイト
  $C：ページサイズは4Kバイト
  $D：ページサイズは8Kバイト
  $E：ページサイズは16Kバイト
  $F：ページサイズは32Kバイト
- IS　論理アドレスのうち上位何ビットを無視するかを指定する
- TIA～TID　論理アドレスの何ビットをテーブルインデックスとして用いるかを決める

TCレジスタのビット配置を図12に示します。

4・3・1 E（イネーブル）ビット

MC68851を動作させるか否かを設定するビットです。‘1’にするとMMUの動作が開始され，アドレス変換機構が働きます。‘0’にするとMMUはアドレス変換動作を行わないようになり，論理アドレスがそのまま物理アドレスになります。

4・3・2 SRE（スーパーバイザルートポインタイネーブル）ビット

通常，MC68851が持っている3つのルートポインタはそれぞれ，DRPがDMAによるアクセス，CRPがユーザモードでのアクセス，SRPがスーパーバイザでのアクセスに使用されます。

SREビットは，スーパーバイザでのアクセスにSRP（スーパーバイザルートポインタ）レジスタを使用するか否かを設定するものです。‘1’を設定すると，スーパーバイザのアクセス（FC2=‘1’のとき）にはSRPが使われるようになります。‘0’にすると，スーパーバイザのアクセスもユーザモードでのアクセスもCRPレジスタが使用されるようになります。

4・3・3 FCL(ファンクションコードルックアップ)ビット

ルートポインタレジスタが指す，最初の変換テーブルのインデックスとしてファンクションコード (FC0～FC2) を使用するか否かを指定します。‘0’を設定するとファンクションコードの使用はディセーブルされ，ルートポインタが指す先にある変換テーブルのインデックスにはISフィールドとTIAフィールドで決まる，論理アドレス中のビットが使用されます。

‘1’を設定すると，ファンクションコードが使用されるようになります。このとき，変換テーブルの数は8個に決まっていますので，ルートポインタレジスタのリミットフィールドは無視されます。

4・3・4 PS(ページサイズ)フィールド

MC68851が，メモリを管理する最小単位であるページの大きさを指定します。PSフィールドの設定値は1ページの大きさがアドレスの何ビット分になるかを示すものとなっています。設定値は$8（1ページは256バイト）から$F（1ページは32Kバイト）までが有効となっており，$7以下の設定は無効です。

4・3・5 IS(初期シフト)フィールド

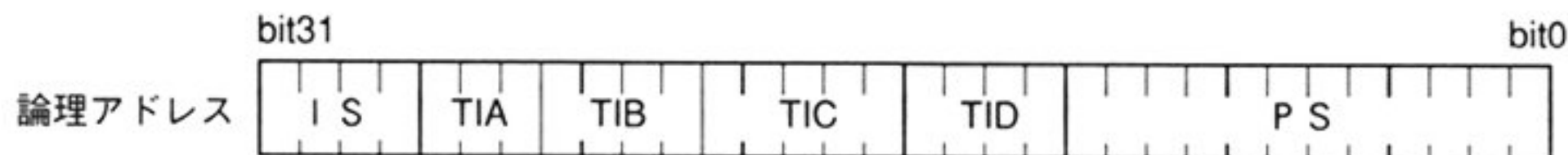
MC68851が持っている論理アドレス入力ピンのうち，上位の何ビットを無視するかを設定します。このビットはアドレスバスが32本未満のCPUにMC68851を接続する場合を考慮して設けられたものです。

MC68020のアドレスバスは32本ありますので，ISフィールドは通常0で使用します。

4・3・6 TIA～TID(テーブルインデックスA～D)フィールド

MC68851が使用する各段の変換テーブルのインデックスとしてアドレスの何ビットを使用するかを指定します。論理アドレスを使ったテーブルの検索では，まず，ISビットが無視され，次のTIA分のビットが使われ，次の段のテーブル検索ではさらにTIB分のビットが使われます。

●図13……ＴＣの設定と論理アドレスの分割の例

TCレジスタの設定：IS=4　TIA=3　TIB=4　TIC=5　TID=4　PS=$C

TIA～TID, IS, PSの各フィールドの設定値は可変ですが，次のような制約があります。

1) TIA＋TIB＋TIC＋TID＋IS＋PS＝32

2) TIA～TIDのうち，先に使用される側が０であった場合，それ以降のフィールドも０でなければならない（たとえば，TIBが０ならTIC，TIDも０）

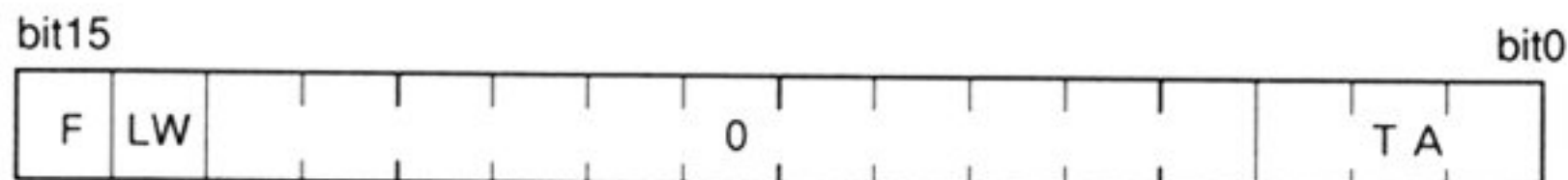
3) TIAは０であってはならない

例として，図13にTCレジスタをIS＝４，TIA＝３，TIB＝４，TIC＝５，TID＝４，PS＝$Cに設定したとき，論理アドレスの各ビットがどのように分割されるかを示します。

4・4 PCSR（PMMUキャッシュステータスレジスタ）

PCSRのPは，MC68851をPMMU（ページメモリ管理ユニット）と呼ぶことからつけられたものです。PCSRは読み出し専用のレジスタで，MC68851が持っているATC（アドレス変換テーブル用キャッシュ）の状態を示すものです。

PCSRのビット配置を図14に示します。

●図14……PCSR（PMMUキャッシュステータスレジスタ）

・F　　０：通常動作
　　　　１：ATCからエントリが削除された

・LW　　０：通常動作
　　　　１：ATCのなかにロックされていないエントリが１つしかない

・TA　　MC68851が内部で管理しているタスクエイリアス値の現在値

4・4・1 F(削除)ビット

MC68851のATC(アドレス変換キャッシュ)には，CRPの変更のたびにMC68851が自動的につけていく「タスクエイリアス」と呼ばれる番号がついています。タスクエイリアスは3ビットあり，最大8種類のCRP値に対応できるようになっていますが，それ以上の変更があった場合には,MC68851は適当なタスクエイリアス番号のものを追い出して新規のものに割り当てます。このとき，追い出されたタスクエイリアス番号のアドレス変換テーブルの内容がATCに残っていると問題がありますので，MC68851はATCからも該当するTA番号を持ったものを削除します。

PCSRのFビットはTAからの削除動作があったことを示すもので，削除動作が行われると‘1’になります。

4・4・2 LW(ロックワーニング)ビット

ATCも通常のキャッシュメモリと同様，新しい内容が入るときに古いものが追い出されます。どれを追い出すかの選択はMC68851が自動的に行いますが，パフォーマンス上の問題などから特定のページだけはどうしても追い出されたくないという場合もあります。これに対応するため，MC68851はATCの置換を禁止（ロック）する機能を持たせています。ページディスクリプタのL（ロック）ビットを‘1’にしておくと，そのディスクリプタがATCに一度入ると，その後は自動的に追い出されないようになります。

ATCの内容をロックすることで，MC68851による追い出しを制限し，パフォーマンスを稼ぐこともできますが，ATCがロックされたエントリばかりになるとMC68851の動作そのものに問題が出てしまいます。

このため，ATCのエントリのなかでロックされていないものが残り1つだけ(すなわち，置き換え可能なエントリが1つしかない状態）になると，MC68851はLWビットを‘1’にします。

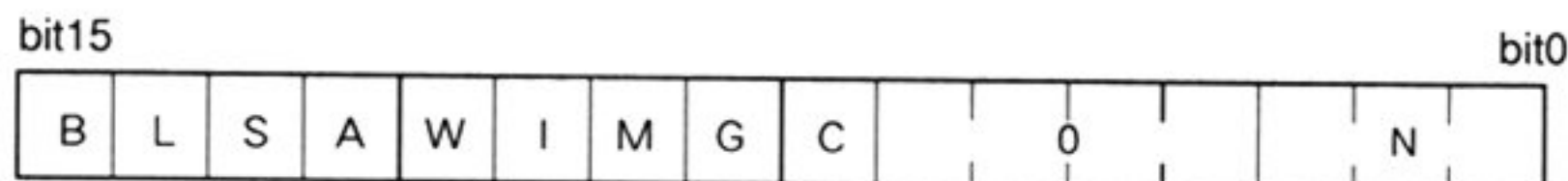
ロック機能を使う場合には，定期的にこのフラグをチェックしておくべきです。

4・4・3 TA(タスクエイリアス)

MC68851が管理しているタスクエイリアス値の現在値を示します。

4・5 PSR(PMMUステータスレジスタ)

PSRは，MC68851の動作中になんらかの障害が発生した場合，その原因を知るために必要な情報を保持するレジスタです。PSRの更新はPTEST命令，PMOVE to PSR命令，PRESTORE命令でのみ行えます。PSRのビット配置を図15に示します。

●図15……PSR（PMMUステータスレジスタ）

4・5・1 B(バスエラー)ビット

レベル1～7を指定したPTEST命令の場合，メモリ上のテーブルサーチを行っている間にバスエラーが返されると，このビットが‘1’になります。レベル0を指定した場合には，一致するディスクリプタがATC内にあり，そのエントリのBERRビットが‘1’になっていると，このビットが‘1’になります。

4・5・2 L(リミット違反)ビット

レベル1～7を指定したPTEST命令の場合，メモリ上のテーブルサーチを行っているときにインデックス値がリミットフィールドの値を超えていることがわかると‘1’になります。レベル0を指定した場合には，このビットはつねに‘0’になります。

4・5・3 S(スーパーバイザ違反)ビット

レベル1～7を指定したPTEST命令を実行したとき，CPUから与えたファンクションコード値がユーザモードでのアクセスを示しており，かつS（スーパーバイザ）ビットが‘1’になったロングフォーマットのディスクリプタが見つかった場合に‘1’になります。レベル0を指定した場合，このビットはつねに‘0’になります。

4・5・4 A(アクセスレベル違反)ビット

レベル1～7を指定したPTEST命令を実行したとき，テストされるアドレスがRAL(リードアクセスレベル)，またはWAL(ライトアクセスレベル)を越えた場合に，このビットが‘1’になります。レベル0を指定した場合には，このビットはつねに‘0’になります。

4・5・5 W(書き込み保護)ビット

PTEST命令を実行した場合，テストしたアドレスが書き込み不可であれば‘1’になります。書き込み不可であるのは，サーチに使用したディスクリプタのなかにWPビットが‘1’になっているものがあった場合か，あるいはアクセスレベル値がRAL以上，またはWAL以上である場合です（アクセスレベル値は大きいほど特権レベルが低くなります）。

4・5・6 I(無効)ビット

レベル1～7を指定したPTEST命令を実行したとき，指定したアドレスが変換できなかった場合に‘1’になります。変換できないと判断されるのは，参照したディスクリプタが無効ディスクリプタであった場合，テーブルサーチを行っているときにバスエラーが発生した場合，リミット違反が検出された場合です。

レベル0を指定したPTEST命令のときには，指定したアドレスがATCに存在しない場合，または存在はするが，そのエントリのBERRビットが‘1’になっているときに‘1’になります。

4·5·7 M(変更)ビット

指定したアドレスに書き込みが行われたことを示します。レベル1～7を指定したPTEST命令の場合には，与えたアドレスに対応するページディスクリプタが存在しており，そのM(変更)ビットが‘1’になっている場合に‘1’になります。レベル0を指定した場合にはATC内に対応するエントリが存在し，そのエントリのMビットが‘1’になっていたときに‘1’になります。

4·5·8 G(ゲート)ビット

指定したアドレスに対応するテーブルのGビットが‘1’になっているときに‘1’になります。Gビットは，そのアドレスが上位レベルにあるサブルーチンを呼び出すときのエントリとして使われることを示すものです。

上位のルーチンを呼び出すときには，MC68020は「モジュールディスクリプタ」と呼ばれるテーブルを使用します。このテーブルには，実際のプログラムのエントリポイントや，データエリアのアドレス，パラメータの引き渡し方法などを指定するようになっています。また，エントリポイントの先頭には，データエリアのアドレスをどのレジスタにセットするかの情報が収められています。

レベル1～7を指定したPTEST命令の場合には，該当するページディスクリプタが存在し，さらにそのGビットが‘1’のときに‘1’になります。レベル0を指定した場合にはATC内に対応するエントリが存在し，かつそのGビットが‘1’になっているときに‘1’になります。

4·5·9 C(グローバル共有)ビット

与えられた論理アドレスに対応するページディスクリプタが存在し，そのディスクリプタ内のSGビットが‘1’になっているときに，このビットが‘1’になります。

4·5·10 N(レベル番号)フィールド

レベル1～7を指定したPTEST命令の場合，アドレス変換に使用したテーブルの段数がセットされます。レベル0を指定した場合には，このビットはつねに‘0’になります。

4・6 アクセスレベル保護用レジスタ

アクセスレベル保護用レジスタには次のものがあります。

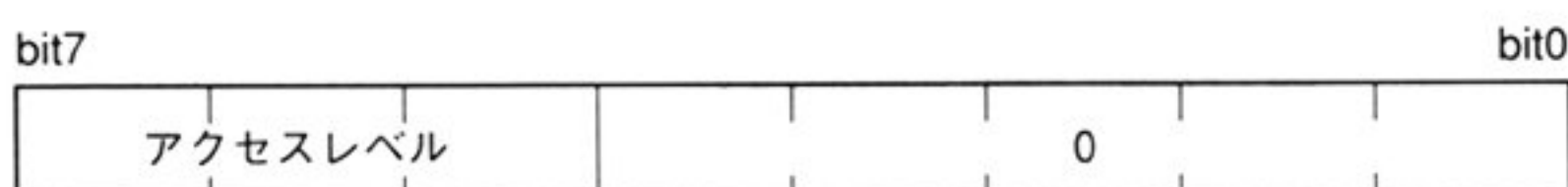
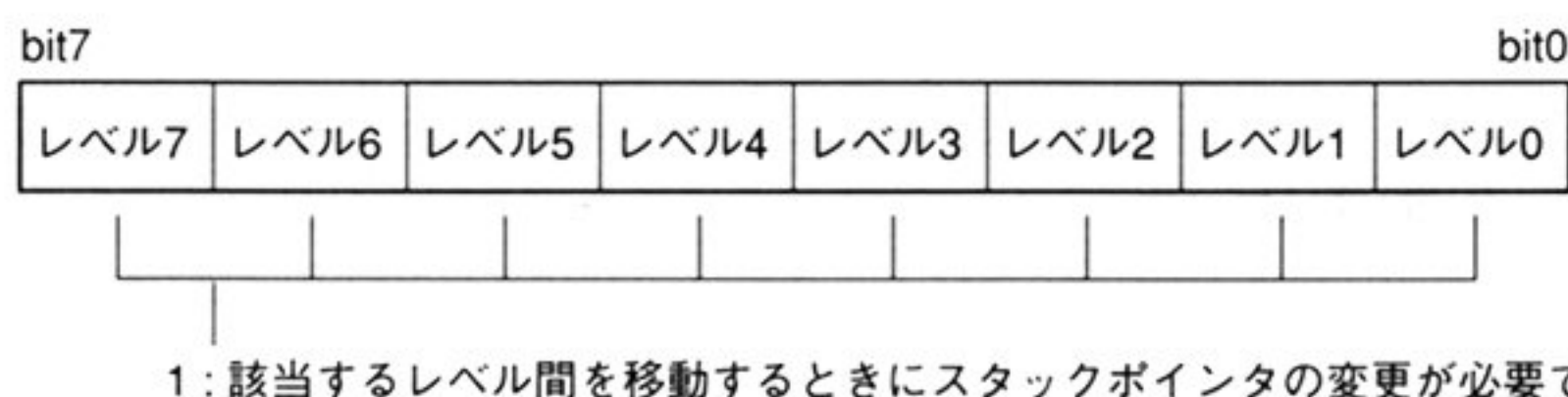
・CAL（カレントアクセスレベル）

・VAL（有効アクセスレベル）

・SCC（スタック変更制御）

このうち，CALとVALのビット配置を図16に，SCCのビット配置を図17に示します。

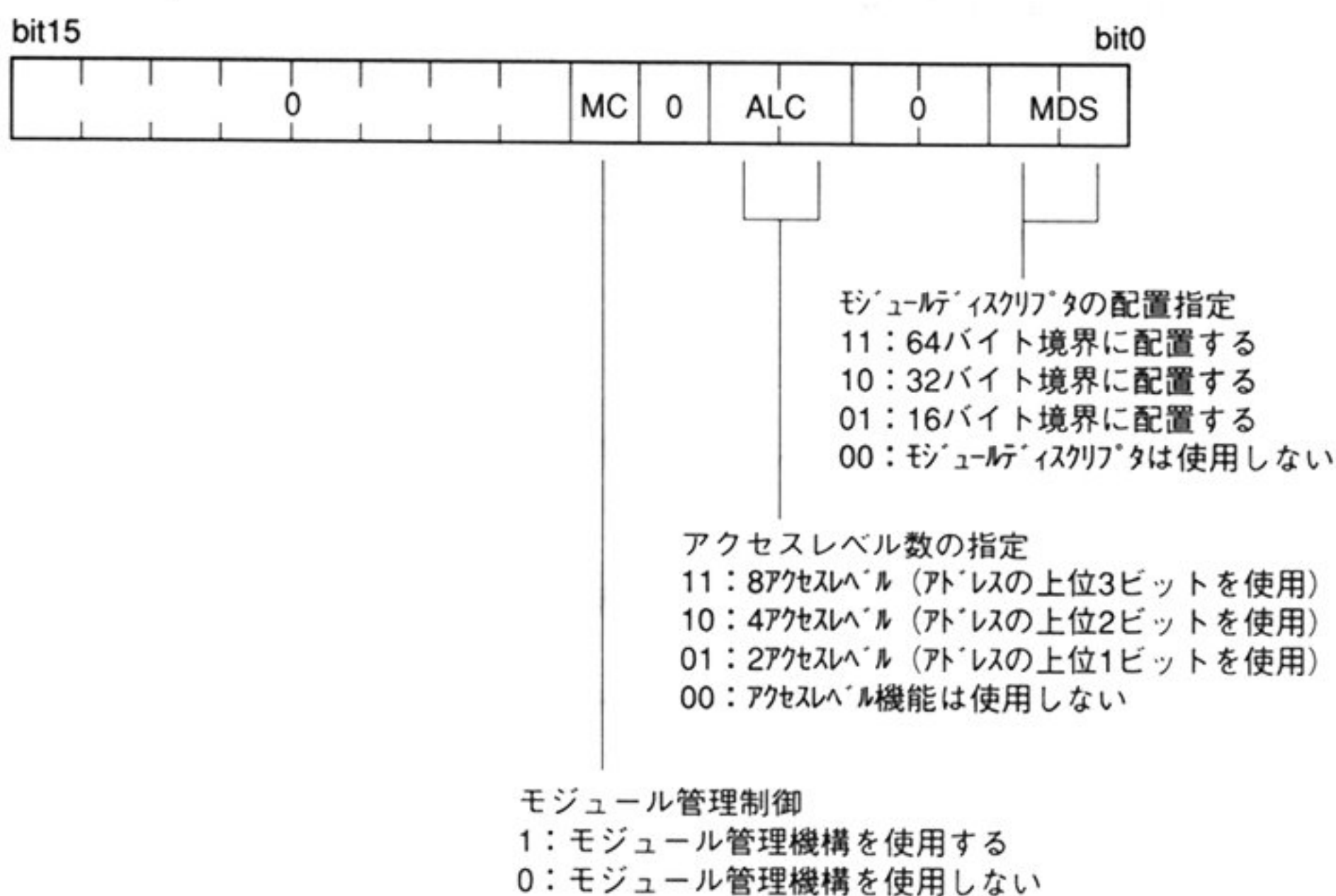
●図16……CAL/VAL（カレントアクセスレベル／有効アクセスレベル）レジスタ

bit7–bit5	bit4–bit0
アクセスレベル	0

●図17……SCC（スタック変更制御）レジスタ

CALは現在のアクセスレベル，VALはレベル間の呼び出しが行われた場合に，呼び出した側のレベルを示しています。

SCCは，CALLM命令によるレベル間移動が行われる場合，CPUに対してスタックの変更を要求する必要があるか否かを指定するものです。たとえばレベル4からレベル2に移動するときには，SCCのビット4～2が検査され，いずれか1つでも‘1’になっているビットがあればスタックポインタを変更するよう，CPUに要求します。

4・7 AC（アクセス制御）レジスタ

AC（アクセス制御）レジスタは，アクセスレベル管理機構の制御を行うものです。ACレジスタのビット配置を図18に示します。

●図18……ＡＣ（アクセス制御）レジスタ

4·7·1 MC（モジュール制御）ビット

MCビットは，MC68851に対しモジュール管理動作を行うか否かを指示するものです。‘1’になっているとモジュール動作がイネーブルされ，CALLM/RTM命令によるレベル間移動の動作が行われるようになります。

‘0’になっているとモジュール管理動作は行われなくなります。CALの変更も行われなくなります。アクセスステータスレジスタ（ALCR）を読んでも$0となり，CALLM/RTM命令はすべてトラップされます。

4·7·2 ALC（アクセスレベル制御）フィールド

アクセスレベルチェック用として，論理アドレスのうち，上位の何ビットをアクセスレベルとして使用するかを指定します。‘00’を指定すると，レベル管理機構は動作しなくなります。

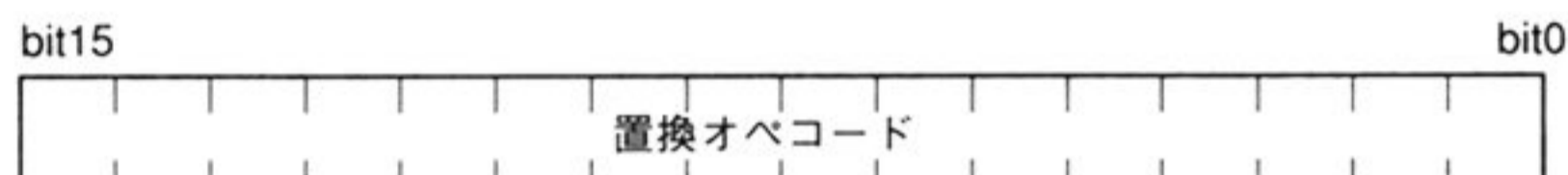
4・7・3 MDS(モジュールディスクリプタサイズ)フィールド

モジュールディスクリプタを何バイト境界に配置するかを指定するビットです。'00'を設定すると，モジュールディスクリプタは配置できなくなります。

4・8 BAD0～BAD7(ブレークポイントアクノリッジデータ)レジスタ

ビット配置を図19に示します。ブレークポイント動作を行っているとき，ブレークポイントアクノリッジサイクルでCPUに渡すオペコードをセットします。ブレークポイント命令は$4848～$484Fの８種類がありますので，BADもそれぞれに対応して８本用意されています。

通常，ブレークポイント動作を行うときには，ブレークをかける場所のオペコードをBADにセットしてからブレークポイント命令と置き換えます。

●図19……BAD（ブレークポイントアクノリッジデータ）レジスタ

4・9 BAC0～BAC7(ブレークポイントアクノリッジ制御)レジスタ

ブレークポイント動作の許可／禁止，ブレークポイントを何回目で例外を発生するかを指定するものです。ビット配置は図20のようになっています。

●図20……BAC（ブレークポイントアクノリッジ制御）レジスタ

bit15
bit0
BPE
0
ﾌﾞﾚｰｸﾎﾟｲﾝﾄｽｷｯﾌﾟｶｳﾝﾄ
何回目のブレークポイント通過で
無効命令とするかを指定
ブレークポイントイネーブル
1：ブレークポイント動作許可
0：　〃　禁止

4·9·1 BPE(ブレークポイントイネーブル)ビット

ブレークポイント命令のそれぞれに対応し，ブレークポイント動作を行うか否かを設定します。'1'を書き込むとブレークポイント動作がイネーブルになります。CPUのブレークポイントアクノリッジサイクル時にBADにセットされた置換オペコードがCPUに引き渡されるようになります。

4·9·2 ブレークポイントスキップカウントフィールド

ブレークポイント命令を何回通過すると例外を発生するかを指定します。該当するブレークポイントアクノリッジサイクルが発生するたびにデクリメントされていきます。スキップカウントが0になっているときにブレークポイントアクノリッジサイクルが検出されると，無効オペコード例外を発生します。

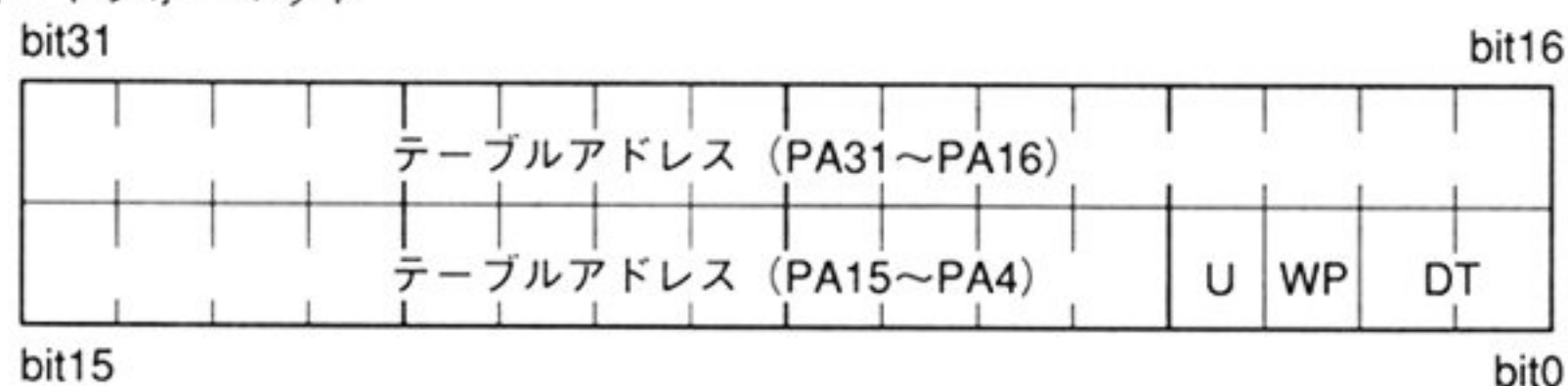
スキップカウンタが0以外のときは，ブレークポイントアクノリッジサイクルではBADにセットされた置換オペコードがCPUに引き渡されます。CPUは置換オペコードを受け取ると何事もなかったように処理を継続します。

これによって，設定したブレークポイント命令がスキップカウントフィールドで指定される回数だけ実行されたときに例外処理を行わせることができます。

5 変換ディスクリプタ

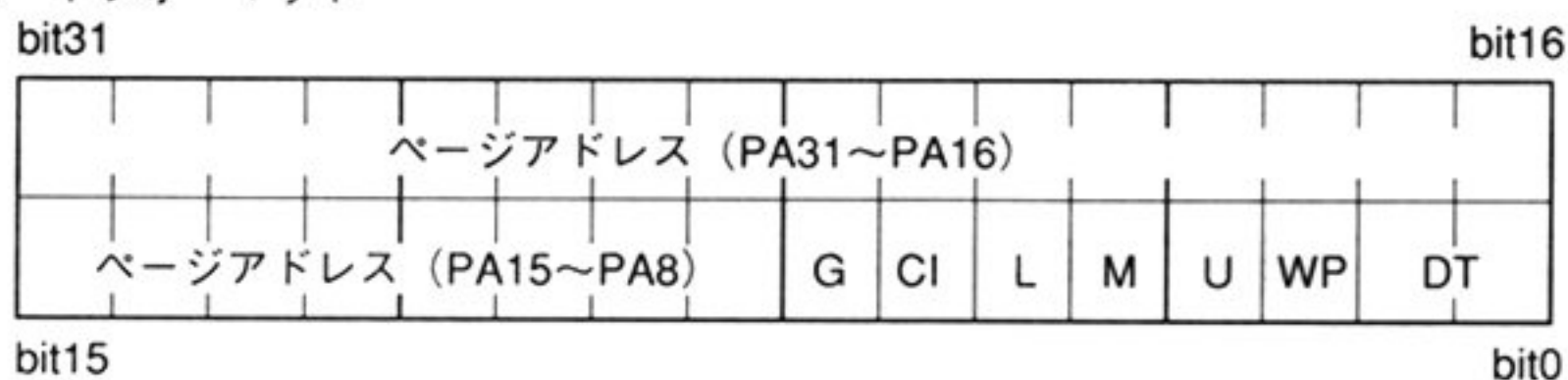
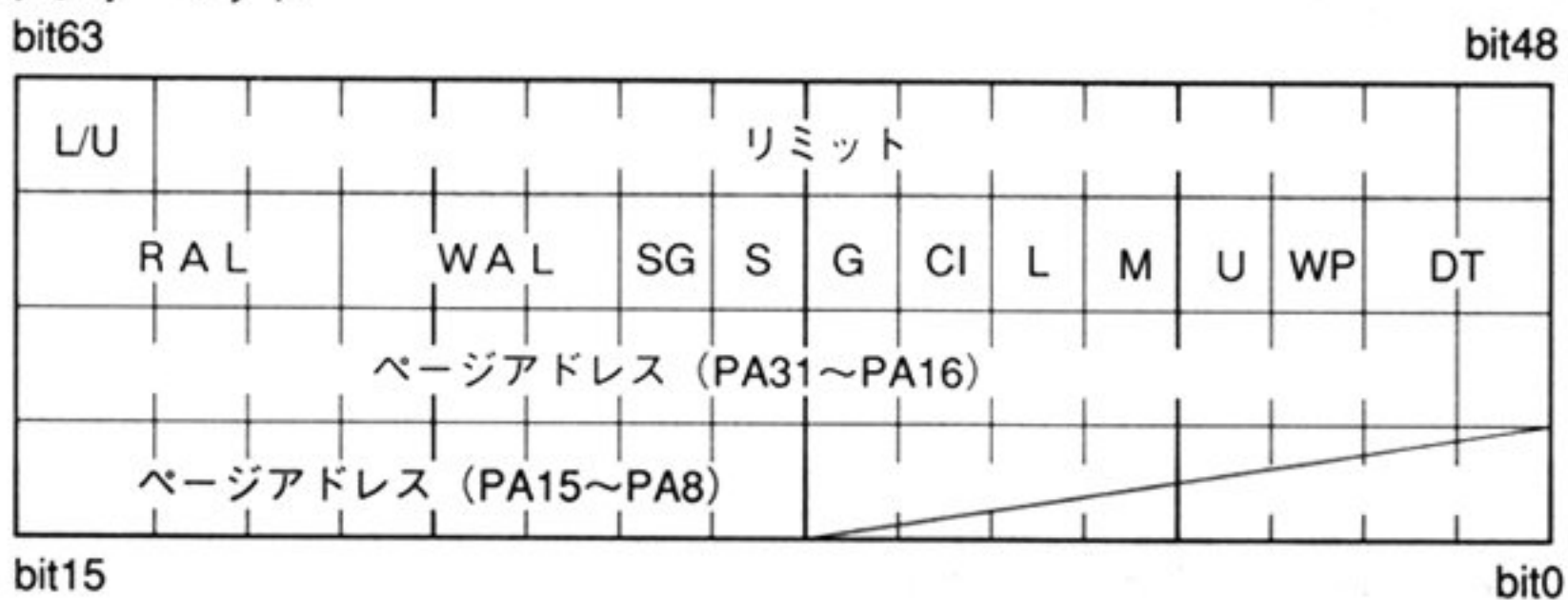
MC68851のページ変換機構が使用する情報の多くは，変換時に参照するディスクリプタ(テーブルディスクリプタ，ページディスクリプタ，間接ディスクリプタ）に記述されています。

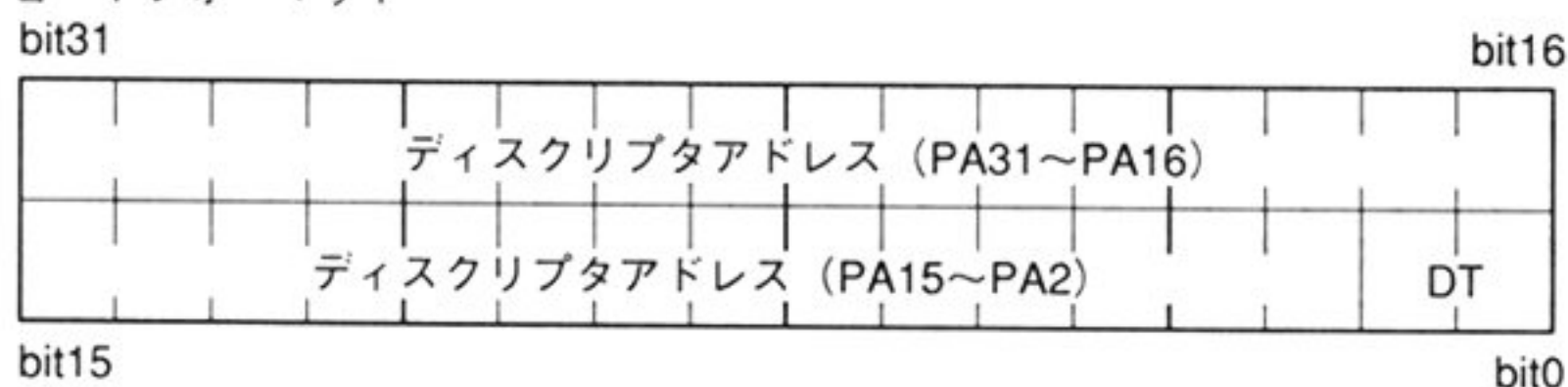
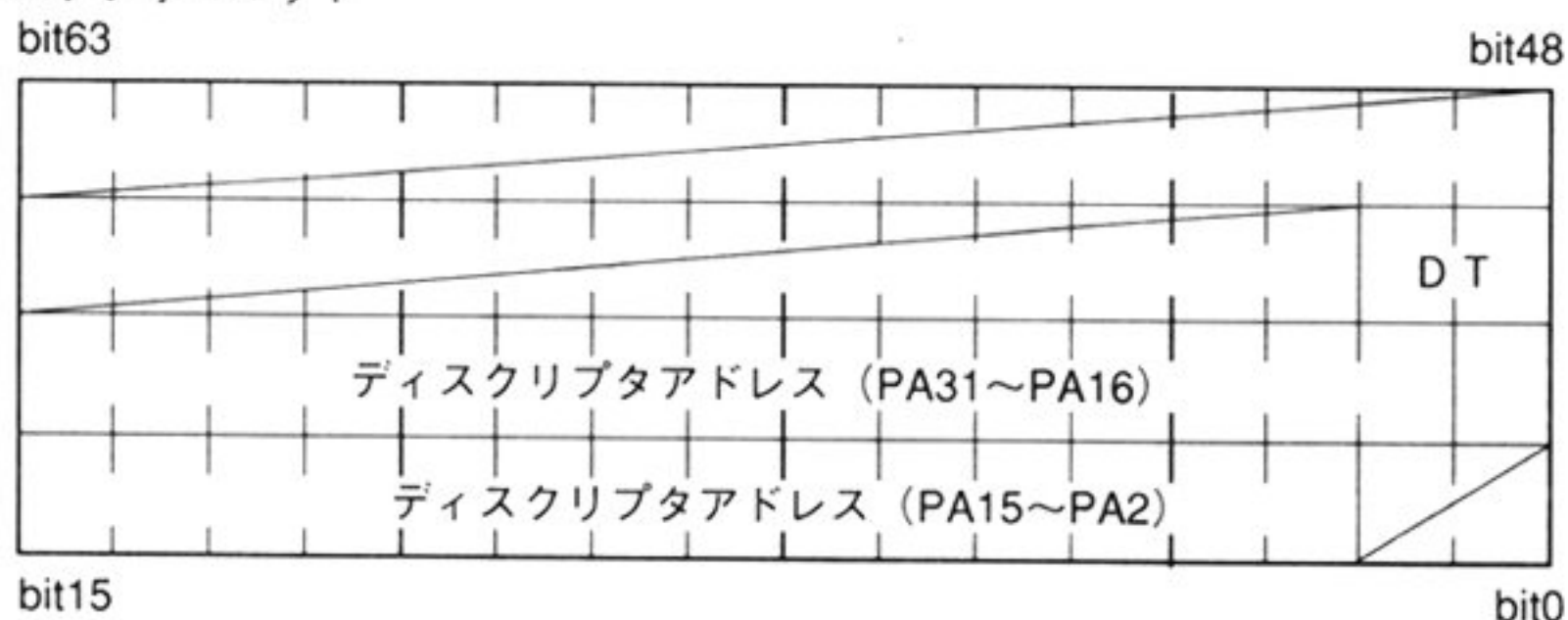
各ディスクリプタは，MC68851が使用する情報をすべて含むロングフォーマットのディスクリプタと，機能を限定してテーブルサイズを節約したショートフォーマットの2種類があります。各ディスクリプタのフォーマットを図21～23に示します。

●図21……テーブルディスクリプタ

ロングフォーマット

＊L/Uビット、およびリミットフィールドはタイプ２の場合のみ有効。

●図22……ページディスクリプタ

ショートフォーマット

ロングフォーマット

＊L/Uビット，およびリミットフィールドはタイプ２の場合のみ有効。

●図23……間接ディスクリプタ

5・1 L/U（上限／下限）ビット

リミットフィールドの値が上限値を示すものであるか，下限値を示すものであるかを指定するものです。‘0’は上限値，‘1’は下限値であることを示します。

5・2 リミットフィールド

テーブルサーチのときに，次段のテーブルをインデックス値の上限や下限を指示するものです。

5・3 RAL（リードアクセスレベル）フィールド

アクセスレベル制御を行っているとき，該当するディスクリプタを使用したリード動作を行えるアクセスレベルを設定します。論理アドレスの上位ビット（アクセスレベルビット）の値

がRAL以下（レベルがRALより上位）でないと読み出し動作は許可されません。

5・4 WAL(ライトアクセスレベル)フィールド

アクセスレベル制御を行っているとき，該当するディスクリプタを使用した書き込み動作を行えるアクセスレベルを設定します。論理アドレスの上位ビット（アクセスレベルビット）の値がRALとWALのうち小さいほう以下（レベルがRALとWALのうち上位のものと同等以上）でないと，書き込み動作は許可されません。

RALやWALは，ショートフォーマットのディスクリプタには存在しません。テーブル検索の途中に1つもロングフォーマットのディスクリプタがない場合には，RALとWALはともに$7，すなわち，どのレベルからもRead/Write可能であるものとみなされます。

5・5 SG(グローバル共有)ビット

当該ディスクリプタで示される論理アドレス空間が全タスクで共有されるか否かを指定するものです。‘1’になっていると共有されることを意味するようになり，ATC内に格納された場合，TA（タスクエイリアス）の一致チェックが行われなくなります。これにより，ATC内でTAが違うだけで内容の同じ変換テーブルが複数格納されるのを防ぐことができます。

5・6 S(スーパーバイザ)ビット

Sビットが‘1’になっていると，そのディスクリプタを使用したアクセス動作はスーパーバイザモードでないと行えなくなります。ユーザモードでもアクセスできるようにするには，このビットを‘0’にしなければなりません。

5・7 G(ゲート)ビット

該当するページにモジュールディスクリプタ（ゲート）が存在することを許すか否かを設定します。‘1’になっていれば，そのページにモジュールディスクリプタを置くことが許可になり，CALLM/RTM命令によるレベル間移動をともなう呼び出し／復帰のために使用できるよ

うになります。

Gビットが'0'になっているページにモジュールディスクリプタは配置できません。

5・8 CI(キャッシュ禁止)ビット

そのページにあるデータをキャッシュメモリ(ATCではなく，通常の命令／データ用のキャッシュメモリ)に格納してもよいか否かを指定するものです。CIビットが'1'になっていると該当するページはキャッシュ禁止となり，アクセスを行ったときにはMC68851はCLI出力をアサートしてCPUや外部回路にキャッシュ動作の禁止を示します。

5・9 L(ロック)ビット

該当するページがATCに格納された場合，そのエントリを追い出さないようにすることをMC68851に要求するビットです。'1'になっていると，そのエントリはATC内でロックされ，MC68851による削除動作が行われなくなります。

ただし，Lビットが'1'になっているエントリであっても，それに対応するルートポインタテーブルエントリを書き換えると削除対象となります。これを避けるためにはSGビットも'1'にする必要があります。

5・10 M(変更)ビット

該当するページに対して書き込み動作が行われたことを示します。'1'になっていれば，そのページは書き込み動作が行われています。このビットはMC68851がセットします。MC68851は，みずからこのビットを'0'にすることはありません。

5・11 U(使用)ビット

該当するディスクリプタが使用されたことがあるかどうかを示します。'1'になっていれば，そのページが使用されたことがあることを示します。アクセスが禁止されているディスクリプタであっても，MC68851によるフェッチが行われた場合には'1'になります。このビットは

MC68851がセットします。

5・12 WP(書き込み保護)ビット

'1'のとき，該当するディスクリプタを使用した書き込み動作が行えないことを示します。このビットが'1'になっていると，アクセスレベルに関係なく，書き込み動作は禁止されます。

あるレベル以下だと書き込み動作が行えないようにするにはWALを使用します。

5・13 DT(ディスクリプタタイプ)フィールド

当該ディスクリプタまたは次の段のテーブルの種別を指定するものです。

・00：無効ディスクリプタ

　当該ディスクリプタは無効であることを示します。

・01：ページディスクリプタ

　当該ディスクリプタがページディスクリプタであることを示します。論理アドレスのすべてのビットを使い切っていればタイプ1，テーブルインデックスに使用していないビットが残っている状態(たとえば，TIA～TIDまで使用しているはずであるにもかかわらず，TIBでインデックスした先にテーブルディスクリプタが読み込まれたような場合)であればタイプ2のディスクリプタです。タイプ2のディスクリプタの場合にはL/U，およびリミットフィールドが有効となります。

・10：有効4バイト

　次段の変換テーブルがショートフォーマットのディスクリプタであることを示します。

・11：有効8バイト

　次段の変換テーブルがロングフォーマットのディスクリプタであることを示します。

5・14 テーブルアドレスフィールド

次段のディスクリプタテーブルの物理アドレスの上位28ビットを保持します。

5・15 ページアドレス

ページの開始アドレスの上位ビットを保持します。ページサイズが256ビットの場合は上位の24ビットを，512バイトなら23ビットをという具合に，ページサイズで指定できる以上のビットを，このフィールドで保持します。

6 MC68030の内蔵MMU

MC68030にはMMU機能が内蔵されましたが，これはMC68851からあまり使われなかったり，CPUに内蔵したことで不要となった機能を削除したものとなっています。

MC68030の内蔵MMUとMC68851の主な違いを次に示します。MC68851のうち，やや高度と思われる機能がごっそり抜け落ちていることがわかります。

1)アクセスレベル管理機構の削除（ACレジスタ削除）

2)ブレークポイント機能の削除（BAD/BACレジスタ削除）

3)ルートポインタテーブルの削除

4)タスクエイリアス機構の削除

5)ディスクリプタのグローバル共有機能の削除

6)ATCのエントリ数を64から22に削減

7)ロック（ATCからの削除禁止）機能の削除

8)透過変換レジスタ（TT0,TT1）の追加

9)DMAルートポインタ（DRP）の削除

10)PCSR（キャッシュステータスレジスタ）の削除

11)命令数の削減

これらのうち，4)，5)は3)にともなって連鎖的に取り払われることになったものです。CRPの変更結果を保存しておくルートポインタテーブルを削除したため，変更結果の管理用につけられたTA（タスクエイリアス）が不要となり，さらにTA管理の欠点を補うために導入されたグローバル共有機能も不要となったわけです。

また，9)はMMUがCPUに内蔵されたことによるものです。MC68030ではMMUがチップ内に入ったため，CPUから出力されるアドレスは物理アドレスしかありません。DMAコントローラも物理アドレスを使用するしかありませんから，DMAの出力したアドレスを変換する

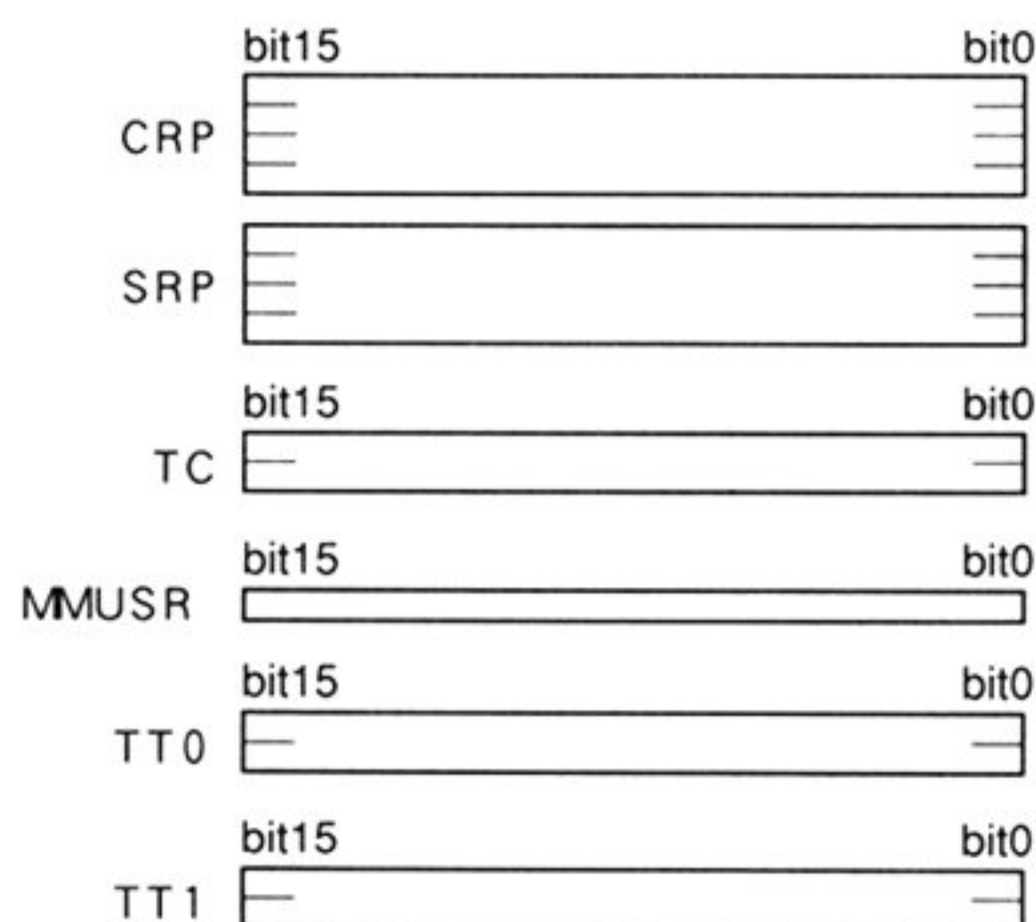
必要もありません。つまり，DRPは不要となるわけです。

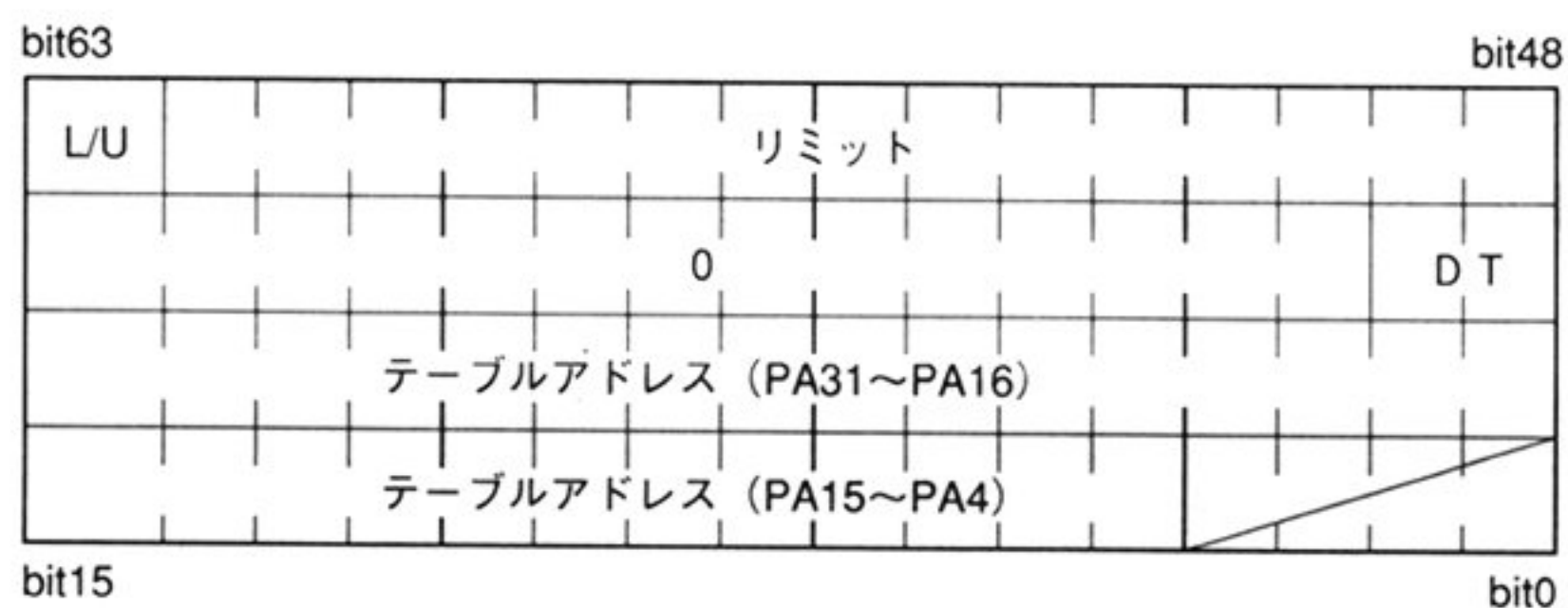
10)のキャッシュステータスレジスタは，MC68851ではロック警告，タスクエイリアスの現在値などを保持していましたが，MC68030ではこれらの機能が削除されたことによりPCSR自体の意味がなくなったため取り払われたものです。

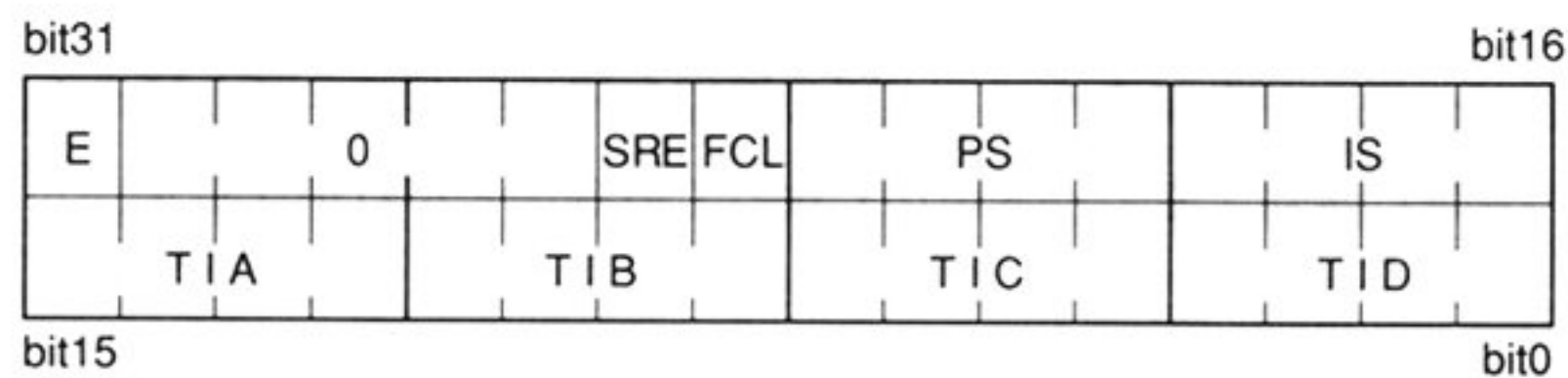
6・1 MC68030のMMU関連レジスタ

MC68030のMMUは，先に示したとおり，MC68851からレベル管理やモジュール管理などの機能を取り払ったサブセット版に相当します。これにともない，レジスタやディスクリプタもMC68851よりも簡略化されています。図24にMC68030の持つMMU関連のレジスタの一覧を示します。

各レジスタのビット配置は図25～28に示してあります。同じ機能を持つレジスタでもアクセスレベルとタスクエイリアスの管理に関係するビットがなくなっていることがわかります。

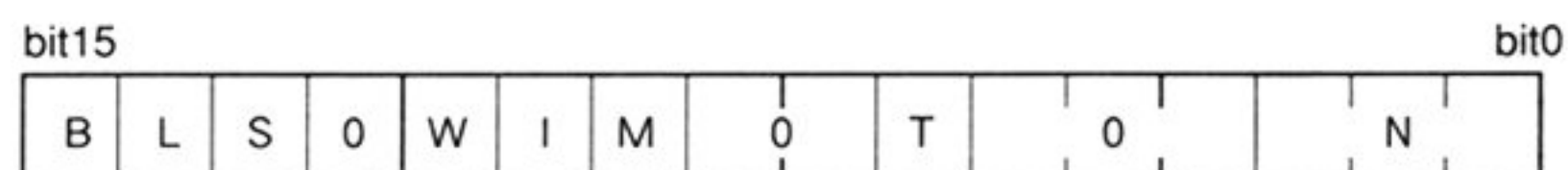
●図24……MC68030のMMU関連レジスタ

・DRPの削除
・PCSRの削除
・BAD0～7の削除，BAC0～7の削除
・ACの削除
・TT0/TT1の追加

●図25……MC68030のSRP/CRP（ﾙｰﾄﾎﾟｲﾝﾀﾚｼﾞｽﾀ）のビット配置

＊SGビットが省略された。

●図26……MC68030のTC（変換制御レジスタ）のビット配置

＊MC68851からの変更はない。

●図27……MC68030のMMUSR（MC68851のPSRに相当）のビット配置

bit15 … bit0

B	L	S	0	W	I	M	0	T	0	N

＊A（アクセスレベル違反）ビット削除
＊G（ゲート）ビット削除
＊C（グローバル共有）ビット削除
＊T（透過アクセス）ビット追加

●図28……MC68030のTT0/TT1（透過変換レジスタ）のビット配置

・論理アドレスベース	論理アドレスのビット31～ビット24との比較値
・論理アドレスマスク	論理アドレスのビット31～ビット24に対するマスク値
・E（イネーブル）	1：透過変換許可　　0：透過変換禁止
・CI	1：キャッシング禁止　　0：キャッシング許可
・R/W	1：リードアクセスが透過　　0：ライトアクセスが透過
・RWM	1：R/Wビットを無視　　0：R/Wビットを使用する
・FCベース	ファンクションコード（FC0～FC2）の比較値
・FCマスク	ファンクションコード（FC0～FC2）に対するマスク値

＊新設されたレジスタ

6・2 透過変換レジスタ(TT0/TT1)

TT0とTT1の透過変換レジスタは新設されたものですので，ここで簡単に説明しておきましょう。

透過変換とは，指定された論理アドレス領域（16Mバイト単位で指定できる）を，保護チェックなどを行わずにそのまま物理アドレスとして出力するものです。TT0とTT1のビット配置を図28に示しています。

6・2・1 E(イネーブル)ビット

透過変換機構を使用するか否かを指定します。‘1’にすると透過変換が許可されます。

6・2・2 CI(キャッシュ禁止)ビット

透過変換を行うブロックのキャッシングを行うか否かを設定します。‘1’にするとキャッシングは禁止になり，透過変換を行ったときにはCIOUT（キャッシュインヒビット出力）端子がアサートされ，外部回路に対してキャッシュ動作を禁止することを指示します。

6・2・3 R/W(リード／ライト)ビット

リード動作に対して透過的とするか，ライト動作に対して透過的とするかを指定します。‘1’にするとリード動作，‘0’にするとライト動作に対して透過的になります。

6・2・4 RWM(リードライトマスク)ビット

R/Wフィールドを使用するか否かを指定します。‘1’にするとリード／ライトのいずれのアクセスに対しても透過変換動作が行われるようになり，‘0’にするとR/Wビットで指定した動作に対してのみ透過的な変換が行われるようになります。

6・2・5 FCマスク(ファンクションコードマスク)フィールド

透過的に変換するときの一致判定用として使用しないファンクションコードのビットを指定します。‘1’を設定したビットは無視されるようになります。‘111’に設定すれば，どのファンクションコードによるアドレッシングであっても透過変換の対象になります。

6・2・6 FCベース(ファンクションコードベース)フィールド

このレジスタに設定したファンクションコードと一致するアクセス動作のみが透過変換の対象となります。ただし，FCマスクフィールドで無視することが指定されているものについては，一致していなくても透過変換の対象となります。

6・2・7 論理アドレスマスクフィールド

透過的に変換するときの一致判定に，論理アドレスのどのビットを使用するかを指定します。‘1’に設定したビットは一致判定の対象外になります。

論理アドレスマスクフィールドは8ビットあり，論理アドレスのビット31からビット24までに対応します。

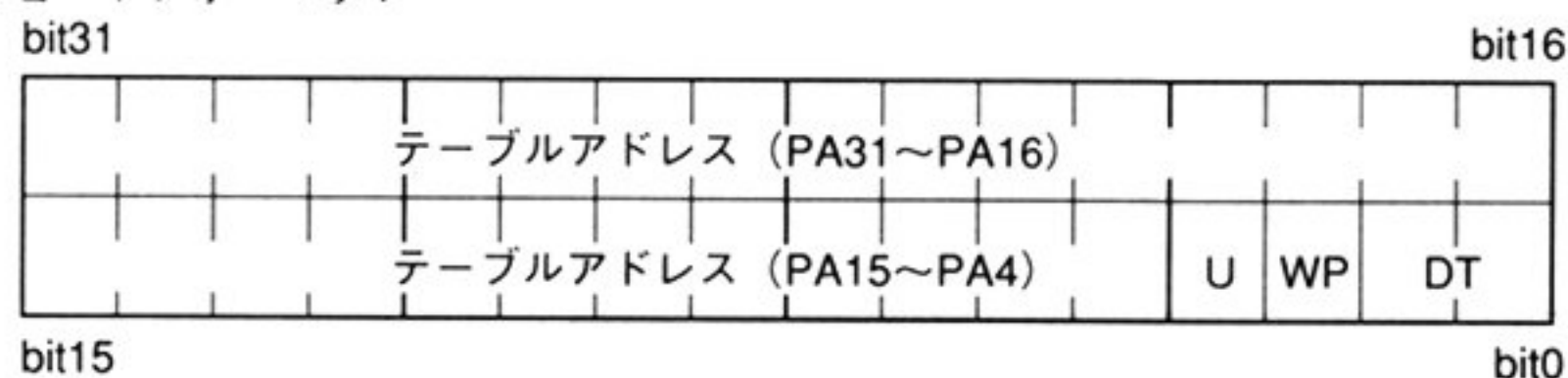
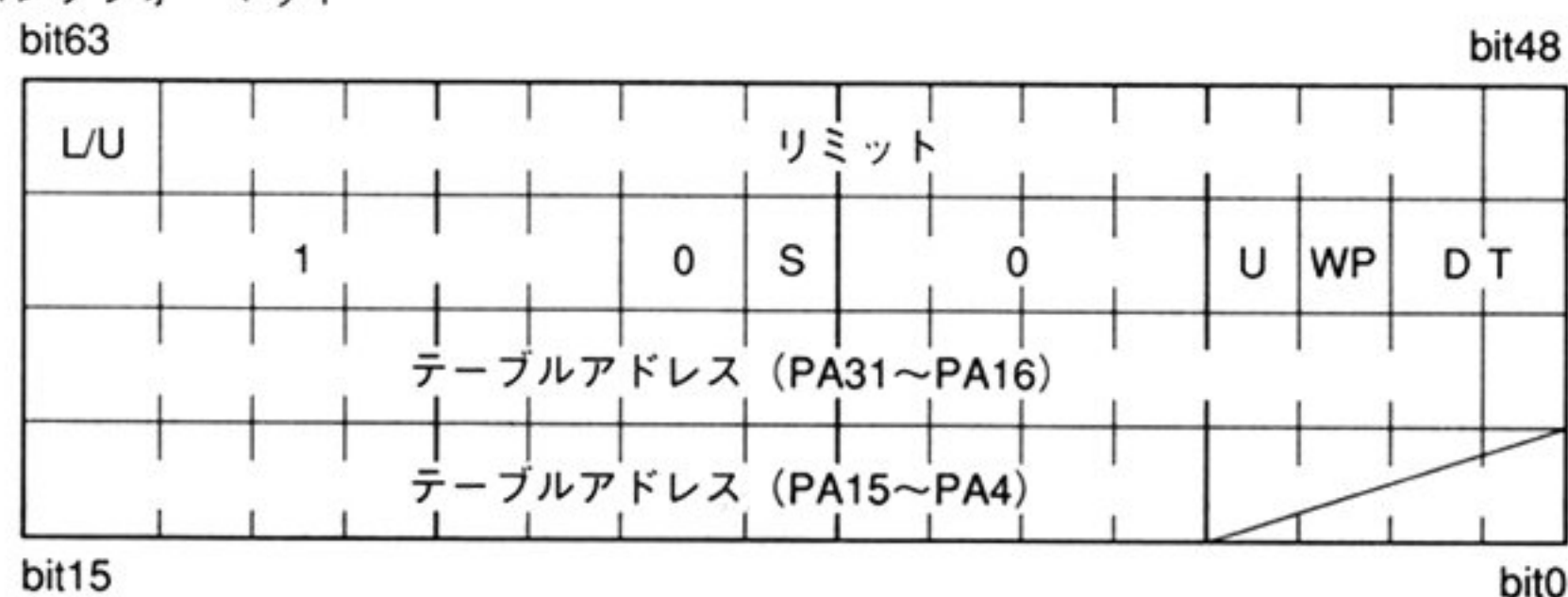
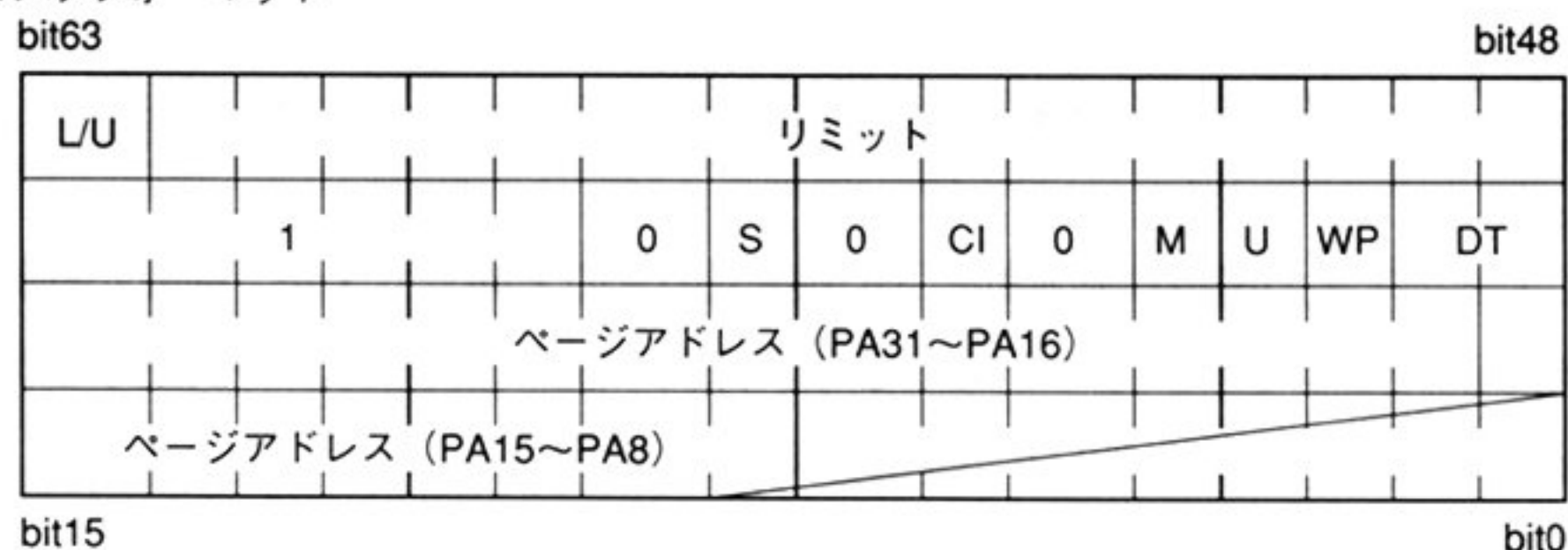
6・2・8 論理アドレスベースフィールド

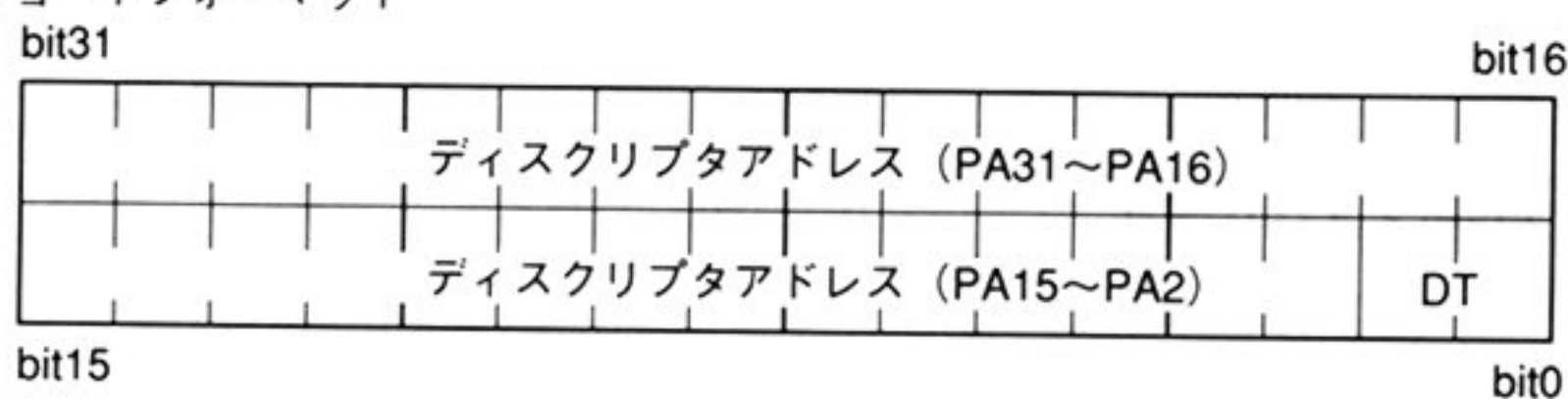
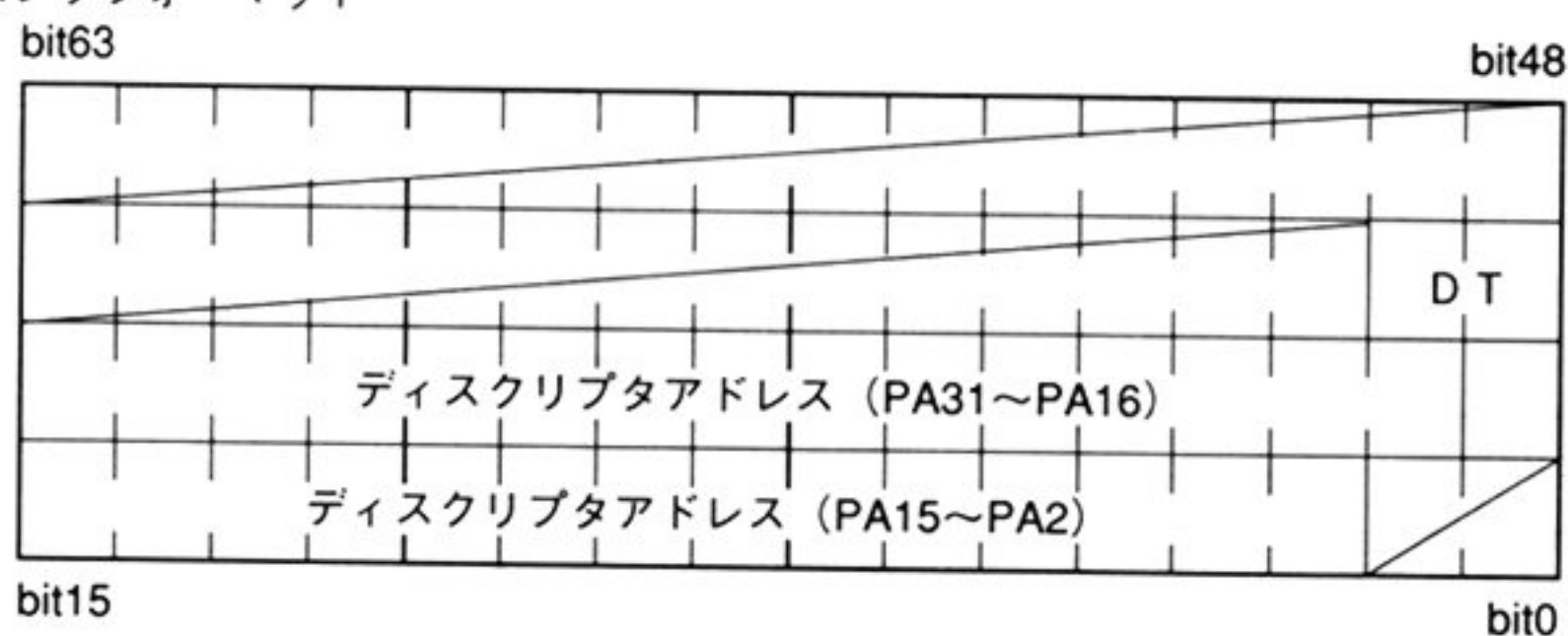
このレジスタに設定された値と論理アドレスのビット31からビット24が比較され，一致したものが透過変換の対象となります。ただし，論理アドレスマスクフィールドで'1'が設定されているビットについては不一致であってもかまいません。

6・3 ディスクリプタの構造

MC68030のディスクリプタの構造は基本的にMC68851のものと同じですが，機能が削減されているために省略されたビットがいくつかあります。図29～31にMC68030が使用するディスクリプタの構造を示します。削除されたビットは次のとおりです。

・SG（グローバル共有）
・RAL（リードアクセスレベル）
・WAL（ライトアクセスレベル）
・G（ゲート）
・L（ロック）

●図29……MC68030のテーブルディスクリプタ

●図30……MC68030のページディスクリプタ

＊L/Uビット，およびリミット・フィールドはタイプ2の場合のみ有効。

●図31……MC68030の間接ディスクリプタ

RALとWALであったビットをすべて'1'(レベル7：最低レベル)，その他はすべて'0'とすることでMC68851を使用したシステムでも動作できるようになっています

7 MC68040の内蔵MMU

MC68040の内蔵MMUの機能を一言でいえば，MC68030のMMUをさらに簡略化したうえで，透過変換レジスタを命令用とデータ用にそれぞれ独立して持たせるようにしたものとなります。

MC68030とMC68040の内蔵MMUの主な違いは次のようになっています。

1)論理アドレス分割，テーブル検索方法の簡略化

・ファンクションコードルックアップ機能の削除

・ISフィールド，TIDフィールド削除

・TIA，TIBフィールドは7ビットに固定

・TICフィールドは5ビットまたは6ビットのみ(ページサイズは4Kバイトか8Kバイト)

・テーブルディスクリプタと，ページディスクリプタの混在禁止

2)ルートポインタレジスタの32ビット化

3)TCの16ビット化

4)MMUSRの32ビット化

5)透過変換レジスタを，データ用，コード用に独立して設定

6)各ディスクリプタは32ビット長（MC68030のショートフォーマットに相当）のみ

1)のアドレス分割方法の簡略化は，MC68040の大きな特徴の1つです。MC68030では(MC68851でも同じ）テーブルを検索する最初の段階でファンクションコードを使用できたほか，論理アドレスを最大6つのフィールド（IS,TIA～TID,PS）に分割し，それぞれを何ビットずつにするかを任意に決められました。MC68040では，ファンクションコードを使用することはできなくなり，さらに論理アドレスの分割は4つ（TIA～TIC,PS）のみ，しかもTIA，TIBは7ビットに固定で，TICとPSの組み合わせは2種類のみ（TIC＝5，PS＝13またはTIC＝6，PS＝12）しか選択できません。

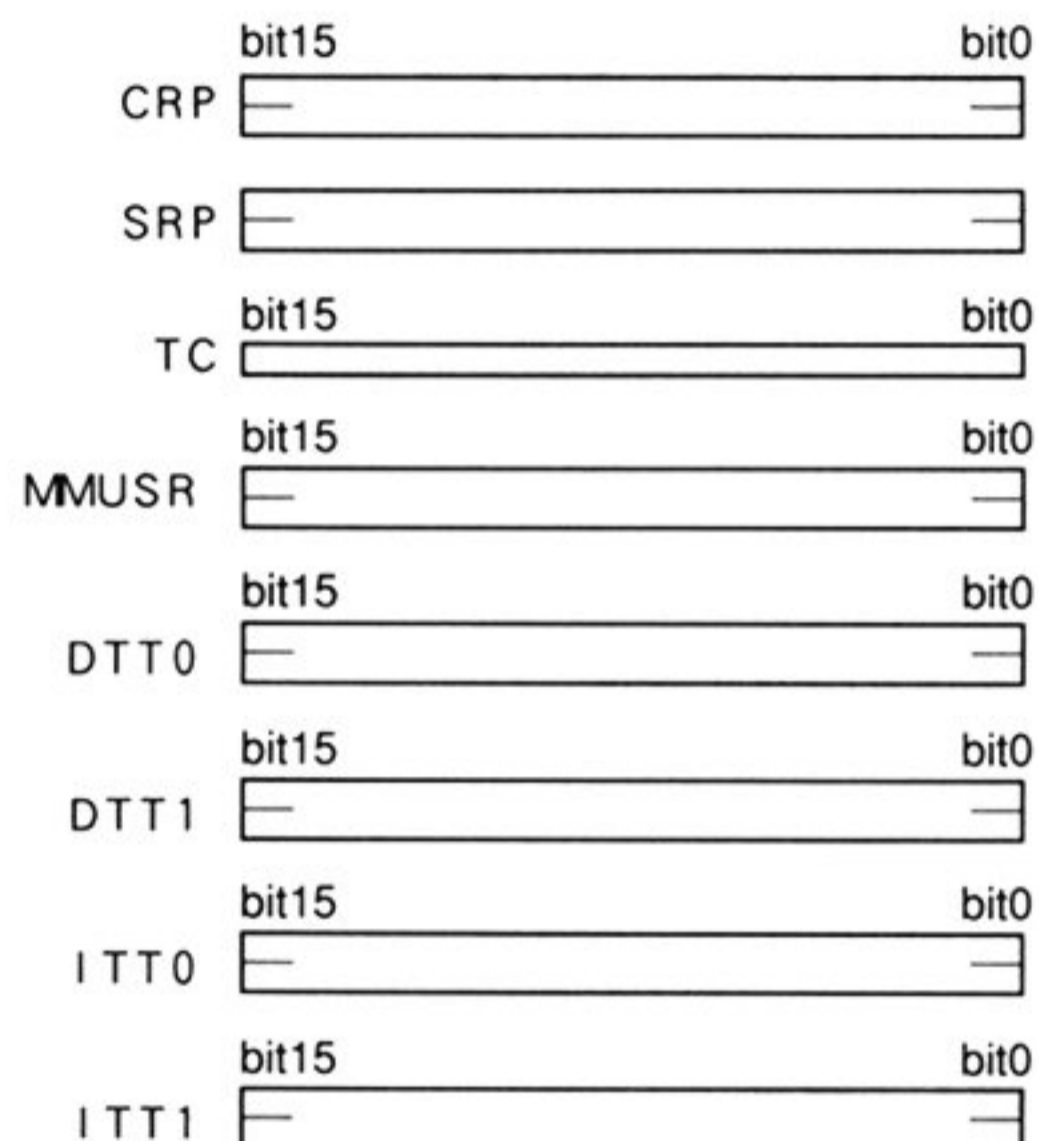
また，MC68030では通常テーブルディスクリプタが存在するところにページディスクリプタを置くことも許されていましたが，MC68040ではこのような方法は使えません。

また，これらの簡略化とともに，各レジスタの見直し，ディスクリプタのフォーマットの見直しなどが行われています。ビット配置などに互換性がない部分も多いので，MC68030用に書かれた仮想記憶管理のプログラムを移植するのはかなり手間がかかると思われます。

7・1 MC68040のMMU関連レジスタ

MC68040のMMU関連レジスタの一覧を図32に示します。MC68030のものと比較すると，レジスタ構成が似て非なるものであることがよくわかると思います。

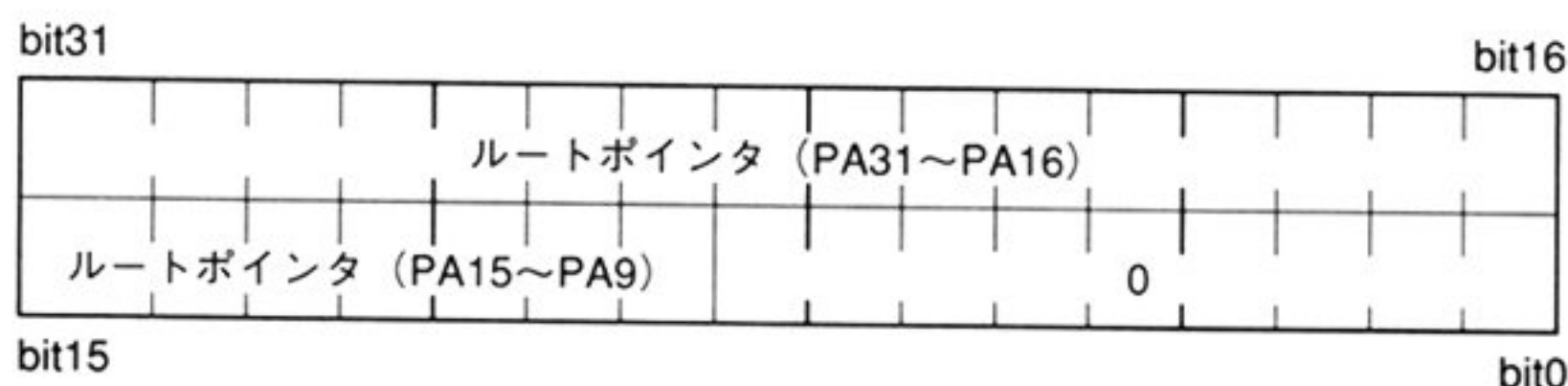
●図32……ＭＣ68040のＭＭＵ関連レジスタ

bit15 bit0
CRP
SRP
bit15 bit0
TC
bit15 bit0
MMUSR
bit15 bit0
DTT0
bit15 bit0
DTT1
bit15 bit0
ITT0
bit15 bit0
ITT1

・ルートポインタレジスタの32ビット化
・TCの16ビット化
・MMUSRの32ビット化
・命令，データそれぞれ独立した透過変換レジスタ（DTT0/1，ITT0/1）

7・2 SRP/URP

MC68040のルートポインタレジスタ（SRP，URP）のビット配置を図33に示します。URP（ユーザルートポインタ）は，MC68030まではCRP（CPUルートポインタ）と呼ばれていたものに相当します。

MC68030と比較すると，ビット長が64ビットから32ビットへと半分になるとともに，L/Uビット，リミットフィールド，DT（ディスクリプタタイプ）が削除されています。MC68030では，ルートポインタレジスタをいきなりページテーブルとして用いるような方法が許されていたほか，ディスクリプタにショートフォーマットとロングフォーマットの２種類があったため，

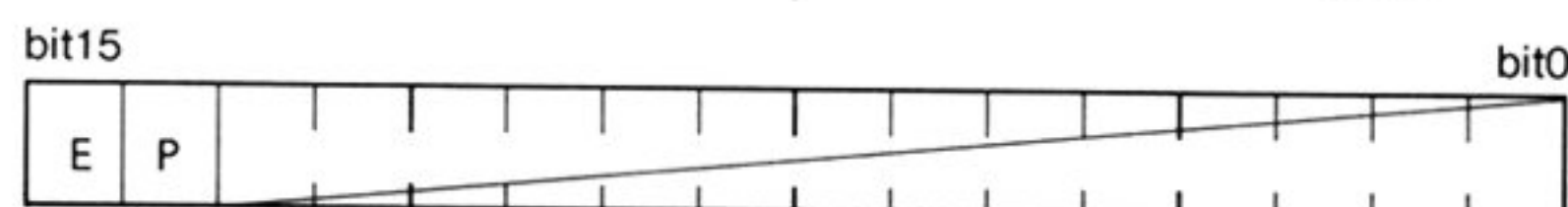
●図33……MC68040のSRP/URP（ルートポインタレジスタ）のビット配置

＊URPはMC68030のCRPに相当
＊32ビット化
＊L/Uビット，リミットフィールド，DTの削除
＊ルートポインタはPA8まで

この識別をDTビットで行っていました。MC68040では，ルートポインタレジスタはあくまでもディスクリプタテーブルを指すものとして使われるようになったのと，ディスクリプタはショートフォーマット（32ビット長）のものだけになったため，DTが不要となりました。また，TIAが7ビットに固定となったことからリミットフィールドなども削除されたと考えられます。

7・3 TC(変換制御)レジスタ

MC68040のTCレジスタのビット配置を図34に示します。変換機構の簡略化にともない，MC68030にはあったフィールド長の指定ビットやFCL，SREビットなどが不要となったため，これらのビットが削除され，残っているのはE(イネーブル)ビットのみとなっています。全体のビット長も32ビットから16ビットに短縮されています。

追加されたP（ページサイズ）ビットは，TIC（およびPS）のフィールド長を選択するもの

●図34……MC68040のTC（変換制御レジスタ）のビット配置

・P（ページサイズ）　1:1ページの大きさは8Kバイト
　　　　　　　　　　0:　　　〃　　　4Kバイト

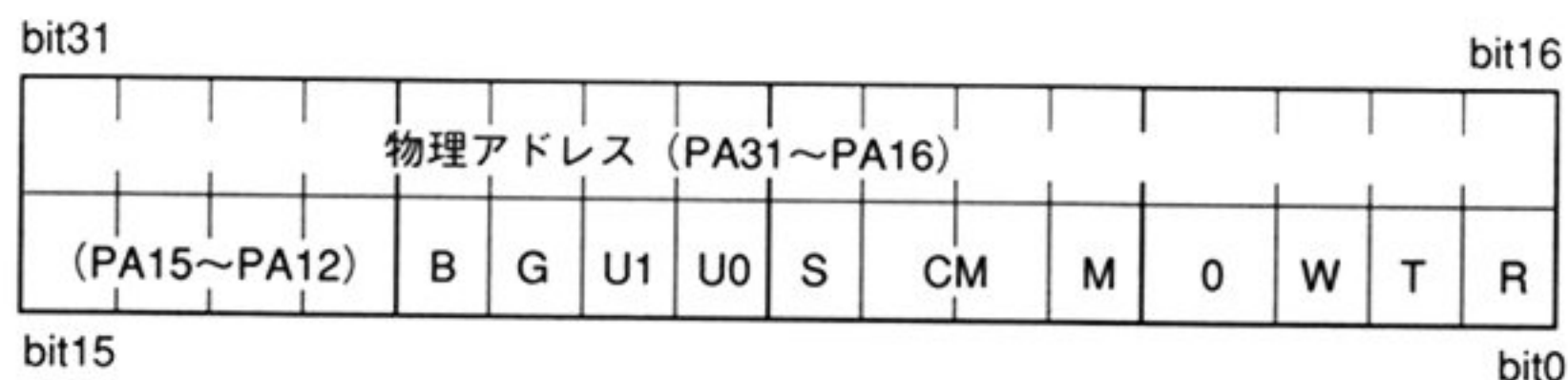
＊P（ページサイズ）ビット追加
＊Eビット以外はすべて削除

です。‘1’になっているとページサイズは8Kバイト（PS=13ビット，TIC=5ビット）となり，‘0’になっているとページサイズは4Kバイト（PS=12ビット，TIC=6ビット）となります。

7・4 MMUSR（MMUステータスレジスタ）

MC68040のMMUSRのビット配置を図35に示します。レジスタのビット長が16ビットから32ビットへと増加したことにともない，PTEST命令で検索されたときの物理アドレスが上位20ビットのフィールドにセットされるようになりました。

また，MC68030と比較するとわかるように，ビット位置や内容に関してかなりの変更が行われています。

●図35……MC68040のMMUSRのビット配置

＊32ビット化，ビット配置全面変更
＊Nフィールド削除
＊Gビット，U1/U0ビット，CMフィールド，Rビット追加

7・4・1 B（バスエラー）ビット

PTEST命令の実行中，テーブル検索時にバスエラーが検出されると‘1’になります。

7・4・2 G（グローバル），U1/U0（ユーザページ属性），S（スーパバイザ保護），CM（キャッシュモード），M（修正済み）

これらのビットには，ページディスクリプタの情報が反映されます。

7・4・3 W(書き込み保護)

検索中にWビットが‘1’になっているディスクリプタがあった場合に‘1’になります（エラーの発生を通知するものではない)。

7・4・4 T(トランスペアレント変換レジスタヒット)

PTEST命令でテストしたアドレスが透過変換レジスタにヒットすると‘1’になります。このとき，Rビットも‘1’になります。

7・4・5 R(常駐)ビット

PTEST命令でテストしたアドレスが透過変換レジスタにヒットするか，あるいは有効なテーブルディスクリプタが検索されてテーブルサーチが終了すると‘1’になります。

7・5 DTT0/DTT1/ITT0/ITT1(透過変換レジスタ)

透過変換レジスタはMC68030では2本だけでしたが，MC68040では命令用，データ用の両方についてそれぞれ2本ずつ，計4本設けられています。

レジスタの内容は，MC68030と比較するとわかるように，下位ワードがかなり変更されています。

●図36……MC68040のDTT0/DTT1/ITT0/ITT1（透過変換レジスタ）のビット配置

＊R/Wビット，RWMビット削除（U1/U0に変更）
＊FCベース，FCマスクフィールド削除
＊Sビット，Wビット追加
＊CMフィールド追加

7·5·1 S(スーパーバイザ／ユーザモード)フィールド

Sフィールドは FC2の使い方を指定します。'00'だと FC='0'(ユーザモードアクセス)のときのみ一致，'01'だと FC='1'（スーパーバイザモードアクセス）のときのみ一致，'10'または'11'のとき FC2は無視されます。

7·5·2 U0/U1(ユーザページ属性)ビット

ユーザで自由に使用できるビットです。

7·5·3 CM(キャッシュモード)フィールド

透過変換が行われる領域でのキャッシュ動作を指定します。

- ・00：キャッシュ許可（ライトスルー動作）
- ・01：キャッシュ許可（コピーバック動作）
- ・10：キャッシュ禁止（シリアライズド）
- ・11：キャッシュ禁止（ノン・シリアライズド）

それぞれの動作については7·6·2を参照してください。

7·5·4 W(書き込み保護)ビット

'1'になっていると，この領域への書き込みが禁止されていることを示します。

7·6 MC68040のディスクリプタの構造

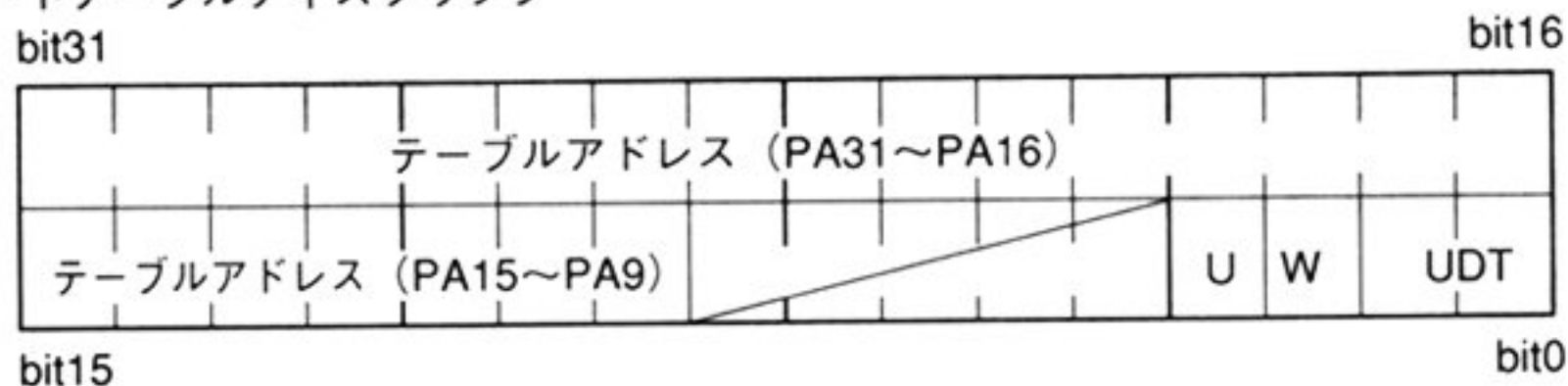
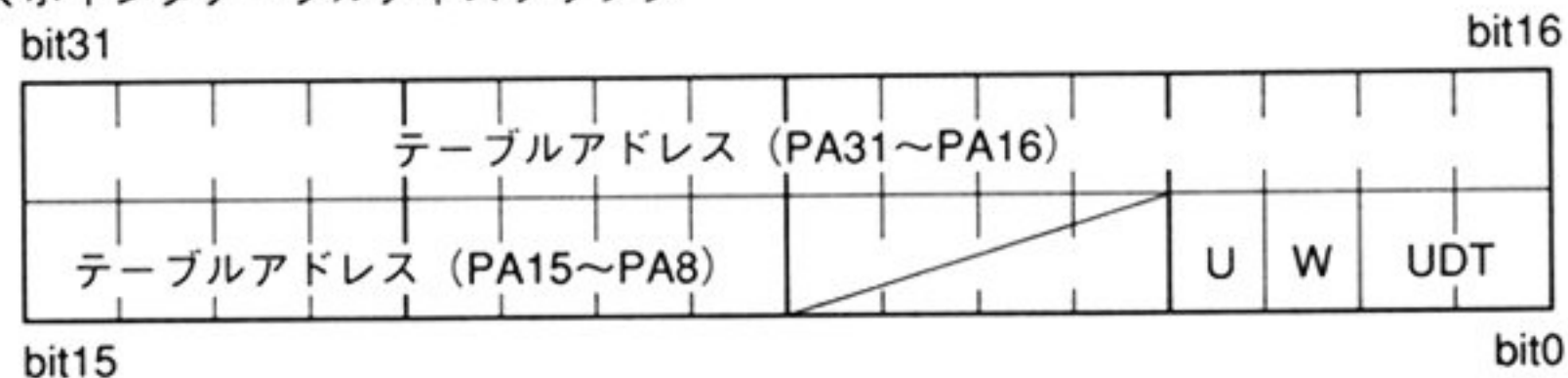
MC68040の使用するディスクリプタの構造を図37〜図39に示します。先に説明したとおり，MC68030では64ビット長のロングフォーマットディスクリプタと，32ビット長のショートフォーマットディスクリプタの２種類がありましたが，MC68040では32ビット長のディスクリプタしかありません。

また，テーブルディスクリプタとページディスクリプタの混在などは許されませんので，ディスクリプタタイプフィールドも最終段でページディスクリプタか，間接ディスクリプタであるかを示す程度のものになっています。

7・6・1 テーブルディスクリプタ

MC68040のテーブルディスクリプタの構造は図37のようになっています。MC68030のショートフォーマットのテーブルディスクリプタとの違いは，CIビットとMビットがなくなっていることと，DTがUDTと名前を変えていることです。

●図37……MC68040のテーブルディスクリプタ

ルートテーブルディスクリプタ

4Kポインタテーブルディスクリプタ

8Kポインタディスクリプタ

＊UDT：上位レベルディスクリプタタイプ

MC68040のUDTはMC68030よりも単純になり，次の段のディスクリプタが存在するか否かを示すだけのものとなっています。'00'か'01'であれば無効，'10'か'11'であればページが存在していることを示します。

7・6・2 ページディスクリプタ

MC68040のページディスクリプタの構造は図38のようになっています。MC68030のものと比べ，かなりビットの入れ替えが行われていることがわかります。

1) URとU1/U0

URはユーザ定義ビットで自由に使用できるもの，U1/U0はユーザが設定可能なページ属性です。URとU1/U0の違いは，MC68040がバスアクセスを行う場合，UPA0とUPA1の信号ピンにU0，U1の設定が反映されるということです。

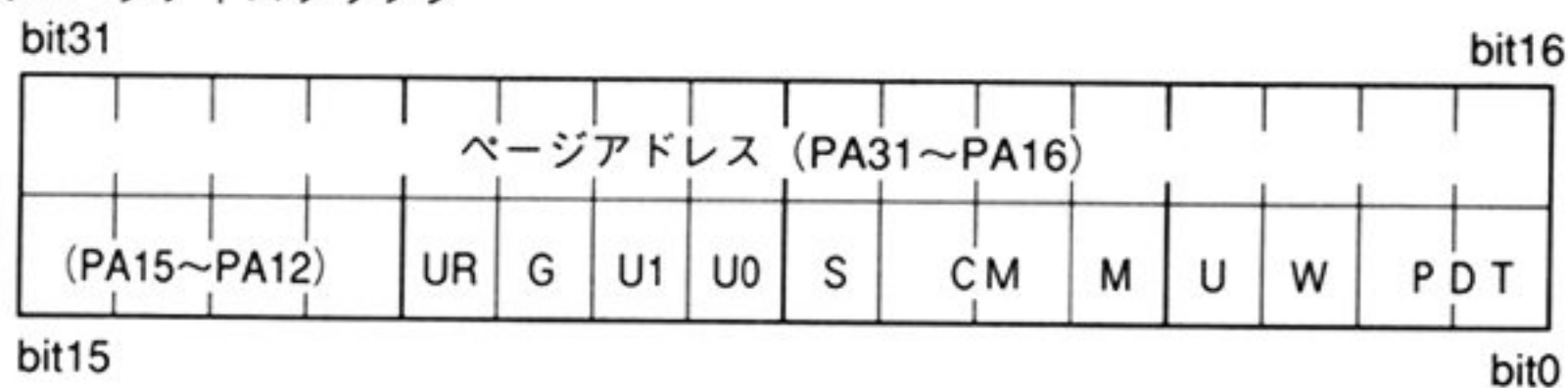
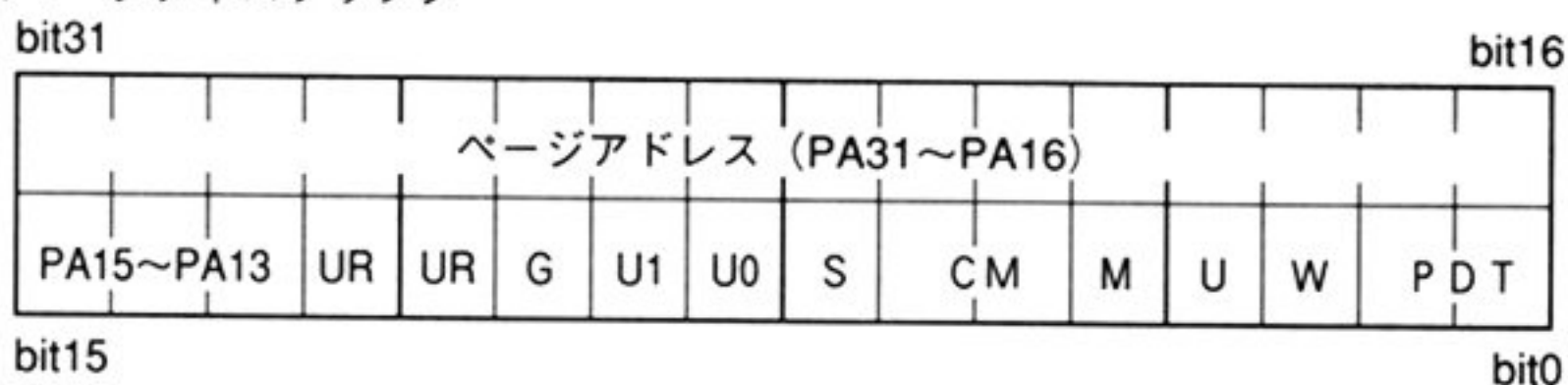
2) G（グローバル）

名前から見ると，MC68851にありながら，MC68030で削除された機能の１つであるグローバル共有の機能が復活したように思えますが，MC68040のGビットはPFLUSH命令でATCエントリから削除されないようにするためのビットとして機能するもので，MC68851のものとは若干異なります。

3) S（スーパーバイザ保護）

このビットが‘1’になっていると，そのページがスーパーバイザアクセス専用であることを示します。ユーザモードでアクセスするとエラーになります。

●図38……MC68040のページディスクリプタ

4Ｋページディスクリプタ

8Ｋページディスクリプタ

＊PDT：ページディスクリプタタイプ
＊UR　：ユーザで任意使用可

＊ＣＩビット削除
＊URビット，Gビット，Sビット追加
＊CMフィールド追加

4)CM（キャッシュモード）

そのページに対するキャッシュ制御回路の動作を指示するビットです。

・00：キャッシュ有効，ライトスルー

キャッシュ動作が行われます。ライト時には必ずメモリへの書き込みを行います。

・01：キャッシュ有効，コピーバック

キャッシュ動作が行われます。ライト時に該当するアドレスの内容がキャッシュメモリに存在していれば，メモリへの書き込みは行わず，キャッシュの内容だけを更新します。

・10：キャッシュ無効，シリアライズド

キャッシュ動作は行いません。外部バスに対してはCIOUT信号がアサートされ，そのアクセスをキャッシングしてはならないことを外部のキャッシュコントローラなどに指示します。外部バスに対するリード／ライトの動作順序は必ずプログラムの順序と一致します。

・11：キャッシュ無効，ノン・シリアライズド

キャッシュ動作は行いません。外部バスに対するリード／ライトの順序は，CPU内部のパイプラインの動作の順序で行われるため，ときとしてプログラム上で期待する順序とは逆転する可能性があります。

5)PDT（ページディスクリプタタイプ）

'00'は無効，'01'と'11'はページが常駐していること，'10'は間接ディスクリプタであることを示します。

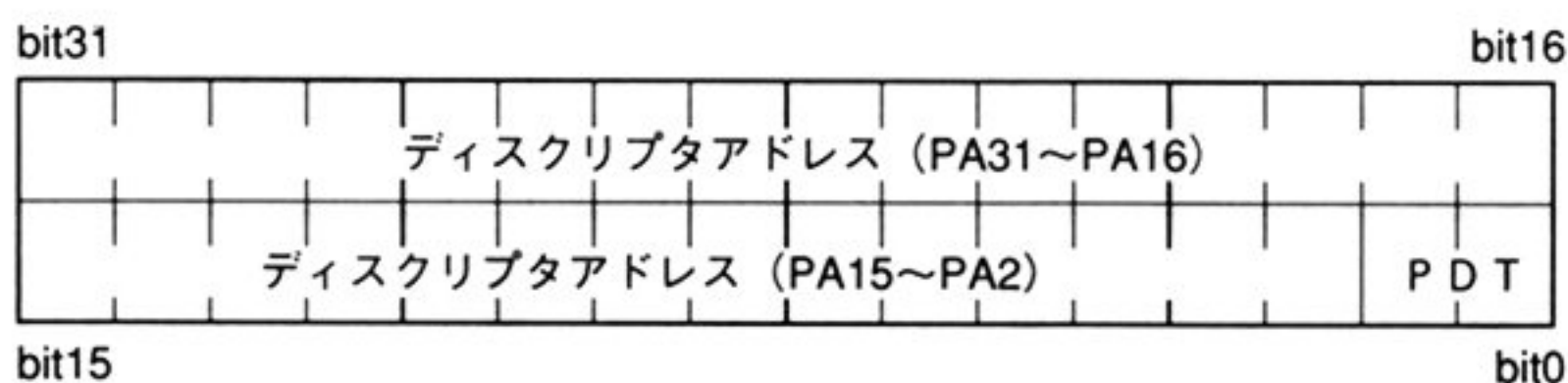
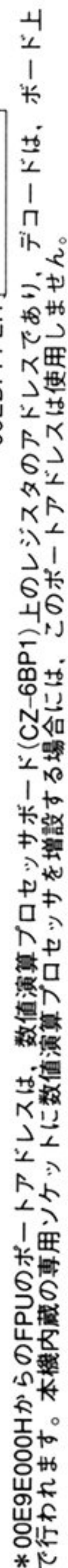
7·6·3 間接ディスクリプタ

MC68040の間接ディスクリプタの構造を図39に示します。MC68030のショートフォーマットの間接ディスクリプタと同じです。PDTはページディスクリプタのものと同じですが，'10'（間接ディスクリプタ）は使用できません。

●図39……MC68040の間接ディスクリプタ

●X68030のメモリマップ

＊00E9E000HからのFPUのポートアドレスは，数値演算プロセッサボード(CZ-6BP1)上のレジスタのアドレスであり，デコードは，ボード上で行われます。本機内蔵の専用ソケットに数値演算プロセッサを増設する場合には，このポートアドレスは使用しません。

あとがき

早いもので，『Inside X68000』が世に出てから2年がたちました。かなり気をつけていたにもかかわらず，あちこちにミスが出てしまったことをこの場を借りてお詫びさせていただきます。

この間，DOS/Vの普及によってPC/AT互換機メーカーが日本に押し寄せ，それにともなって多くの海外のソフトウェアやハードウェアが流れ込んできています。パソコン雑誌も多数創刊され，それらの雑誌ではサウンドボードとCD-ROMを取り付け，Video for Windowsを乗せただけの，自称「マルチメディアマシン」や，CPUとクロック周波数をとっかえひっかえしただけの自称「高性能機」が幅をきかせているようです。

これらの機械が「世界共通」を掲げており，またPC/AT互換機関係の雑誌や単行本が数多く登場したことから，ハードウェアの解説記事や『Inside X68000』的な本が出てくるものと期待していたのですが，いまだに見あたりません。

もっとも，これにはPC/AT互換機特有の事情があるためともいえます。音声も画像もSCSIインターフェースさえも拡張ボードであるがゆえに，次々に性能向上が行われているPC/AT互換機ですが，逆にそうなってしまっているがゆえにハードウェア仕様はより完全なブラックボックスとなってしまっています。ボードを買ってきても，マニュアルに載っているのはインストール方法の説明ばかりで，ハードウェア仕様に関する説明は何もありません。

しかも，本体ごとにハードウェア仕様が異なるため，ハードウェアを直接操作するようなプログラムが他人のマシンで動くことは期待できません。プログラムのやりとりを考えると，BIOSなり，システムコールなりを利用したプログラムにするよりほかに方法がないわけです。

やはり，X68000/X68030シリーズのように，ユーザが直接CRTCの画面モード設定をいじってみたり，VRAMやDMA，SCCなどへの直接アクセスや，ジョイスティックポートへ周辺機器の接続を行うようなことがさほど特殊なことでもなく行われるというのは珍しいことになっていくのでしょうか。

確かに，ハードウェアを自作するのはもちろん，ハードウェアに直接アクセスするようなソフトウェアを作ることですら禁断のテクニックであるかのようにいわれることの多いのが最近の傾向です。

しかし，よく考えてみると妙な話です。仕事で作るアプリケーションプログラムならばいざ知らず，個人が趣味で使うパーソナルコンピュータ（パーソナルワークステーションを含む）

は基本的に何を行ってもよいはずです。ハードウェアにアクセスするのを御法度であるかのようにいっていますが，割り込みハンドラやデバイスドライバはハードウェアをアクセスしているのです。

時計やラジオなどを拾ってきては修理してみたり，バラして外した部品で別のものを組み立ててみたりと，機械の中の構造や仕組みに興味を覚えながらだんだん詳しくなっていくのはごく基本的なアプローチではないでしょうか。

本書の執筆のため，X68030を購入し，翌日には分解して電線で信号を引き出し，レジスタへの設定値などをいじりながら，ロジックアナライザやシンクロなどで波形をとり，電卓を片手にあぁでもない，こうでもないと，計算式を割り出したりしていました。その最中，テスタ1つでアンプやラジオの組み立てたり，拾ってきたテレビを修理したり，改造してディスプレイ代わりに使っていたときのことをふと思い出してしまいました。

あの頃はよかったなどというほどの月日がたったはずではないのですが，この間にすっかり電子機器がユーザからは見えないものになってしまったことを実感しました。以前はテレビやステレオを買えば必ず回路図がついてきたものですが，最近はサービスセンタの一覧がついてくる程度です。パソコンの回路図が公開されることも少なくなりました。

X68000/X68030シリーズは，国産のパソコン技術の結晶です。浜辺の砂のごとくばらまかれたパソコン全体から見ればごく小さなものにすぎませんが，しっかりとした輝きを持った結晶です。みなさんがこの結晶の輝きのなかから何かをつかもうとしたとき，本書がなんらかの手助けとなれば，これに勝る喜びはありません。

1994年1月20日　　　　桒野雅彦

参考文献

●MC68040 USER'S MANUAL	モトローラ
●MC68040ユーザーズ・マニュアル	モトローラ
●MC68030 USER'S MANUAL	モトローラ
●MC68020ユーザーズ・マニュアル	モトローラ
●M68000 FAMILY REFERENCE	モトローラ
●MC68881 USER'S MANUAL	モトローラ
●32ビットマイクロプロセッサTLCS-68000（TMP68030）	東芝
●16/32ビットマイクロプロセッサTLCS-68000周辺デバイス	東芝
●16/32ビットマイクロプロセッサTLCS-68000周辺デバイス	東芝
●日立ICメモリ3（DRAM，DRAMモジュール）	日立

使用測定機類

●マルチメータ	サンワ	CP-7D
●デジタルマルチメータ	FLUKE	75
●ロジックアナライザ	テレシステムズ	PA-200
●シンクロスコープ	ケンウッド	CS-5130

INDEX ……記号・数字・英文

N

R

S

INDEX ……和文

あ

か

さ

た

な

は

ら

X68030 Inside/Out

1994年3月30日初版第1刷発行

編　者　桑野雅彦（くわの まさひこ）

発行者　橋本　五郎

発行所　ソフトバンク株式会社　出版事業部

〒103　東京都中央区日本橋浜町3-42-3

販売　☎ 03(5642)8101

編集　☎ 03(5642)8140

印刷所　株式会社厚徳社

Printed in Japan 1994.

ISBN4-89052-499-1 C0055

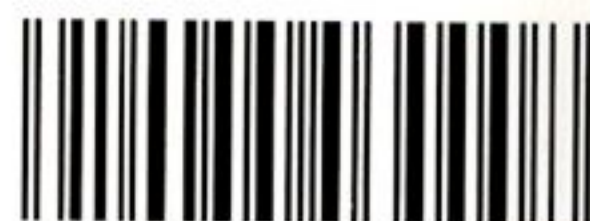
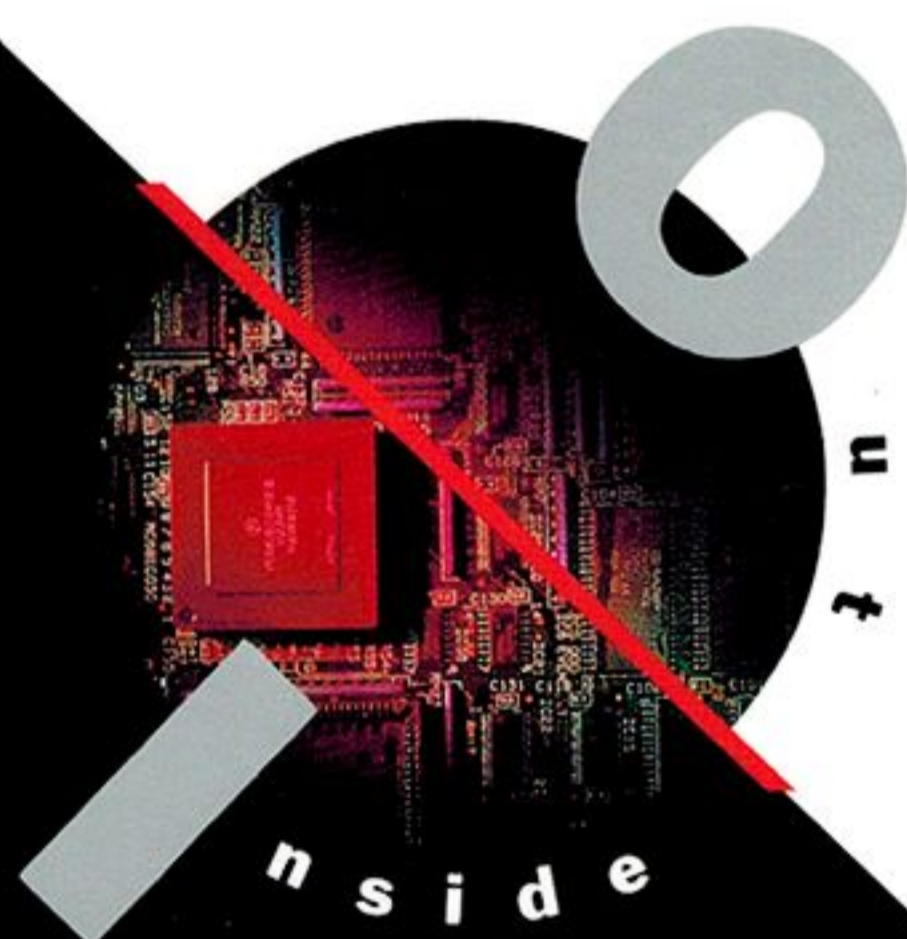
落丁、乱丁本は小社販売局にてお取り替え致します。

定価はカバーに表示してあります。

X680
回路図
1. メイン基板
メイン基本配線図(1)
メイン基本配線図
2. メイン基本配線
メイン基本配

ISBN4-89052-499-1

C0055 P3000E

定価3,000円
（本体2,913円）

X68030

X68000と比較しながら、

X68030のハードウェアの特徴を、

本体内部と外部にわたって解析した本。

さらに、『Inside X68030』執筆後、

筆者が見つけた機能についても

補足的に説明してあります。

また、X68030にはついていなかった

MMU機能についての解説もあります。

X68030の回路図付き。